现代应用数学丛书

计算技术

〔日〕高桥秀俊 著

上海科学技术出版社

现代应用数学丛书

計算技术

〔日〕高橋秀俊 著
姚 晋 譯
郑守淇 校



上海科学技术出版社

內 容 提 要

本书是日本岩波书店出版的現代应用数学丛书之一的中譯本，主要介紹有关計算技术方面的一些基础知識。全书計分“数字計算机的程序設計”和“邏輯綫路和自动机理論”两編，可供計算数学和計算技术工作者参考。

原书分两册出版，現合并成一册。

計算機械 I. II.

〔日〕高 橋 秀 俊

岩波書店 1958

現代应用数学丛书

計 算 技 术

姚 晋 譯 郑守淇 校

上海科学技术出版社出版（上海瑞金二路 450 号）

上海市书刊出版业营业許可証出 093 号

上海市印刷六厂印刷 新华书店上海发行所发行

开本 850×1156 1/32 印張 4 18/32 排版字数 106,000

1965 年 8 月第 1 版 1965 年 8 月第 1 次印刷

印数 1—2,000

統一书号 13119·661 定价(科六) 0.70 元

出版說明

这一套书是根据日本岩波书店出版的"現代应用数学讲座"翻譯而成。日文原书共15卷60册,分成A、B两組,各編有序号。現在把原来同一題目分成两册或三册的加以合并,整理成42种,不另分組編号,陆續翻譯出版。

这套书涉及的面很广,其內容都和現代科学技术密切有关,有一定参考价值。书中收集的資料都比較丰富,而敍述扼要,篇幅不多,有利于讀者以較短时間掌握有关学科的主要內容。虽然,这套书的某些观点不尽适合于我国的情况,但其方法可供参考。因此,翻譯出版这一套书,对我国科学技术工作者可能是有所助益的。

由于日文原书是1957年起以讲座形式陆續出版的,写作时間和篇幅的限制不可避免地会影响原作者对內容的处理,为了尽可能地减少这种影响,我們在某些譯本中,特請譯者或校閱者撰写序或后記,以介紹有关学科的最近发展状况,并对全书內容作一些評价,提出一些看法,結合我国情况补充一些資料文献,在文內过于簡略或不足的地方添加了必要的注釋和改正原书中存在的一些錯誤。希望这些工作能对讀者有所帮助。

欢迎讀者对本书提出批評和意見。

上海科学技术出版社

08070

目　录

第一編　数字計算机的程序設計

§1 序　　言

今天使用的計算机，大致分为**数字型**和**模拟型**两种。数字型計算机是由算盘发展而来的，其特点为：

（i）数是不連續的，只能以整数进行計算。

（ii）大的整数用多位数（例如十进制），即以若干位整数的組合表示。

模拟計算机可看作是从計算尺发展起来的，利用物理量来表示数（例如长度、电阻、电压等），根据物理量間的物理法則进行計算，数是取連續变化的。

模拟型計算机发展較早，从利用面积仪制成的各种积分器，調和解析机等精巧而广泛使用的机器，以至最近大規模的微分分析机（解微分方程的机器），以及利用电气网絡組成的各种解联立一次方程組和微分方程的装置等，都属此类型。

模拟計算机比較适用于某些希望迅速求解的問題，但解答的精确度是不高的（約为 $10^{-2}\sim10^{-3}$）．精确度要求高的計算，非用数值計算不可。因此，数字型的手搖式或电动台式計算机就得到广泛的应用。这是一种沒有特定目的的通用計算机。要提高精确度就要增加位数，但必要的位数是和精度的对数成比例的，因此不会因位数太多而成問題。

随着精密計算需要量的增长，以及冗长計算中誤差积累对計算的影响，开始了数字型計算机的高速化和自动化問題的研究。之后，Harvard 大学制成了巨大的自动計算机 MK-I，MK-II．在高

速化方面，Pennsylvania 大学以几万个电子管制成了电子計算机 ENIAC，它具有較高的計算速度。其后，各国爭相研究，到現在电子計算机已經成为很普通的計算工具了。

数字計算机与模拟計算机比較，最大的特点是能用于多种目的（万能性），容易得到所希望的精确度，計算是明确的（不是說沒有誤差，而是說重复若干次解題，能得出相同的解答，而不依賴于眼睛的观察）。它的本质的特点是具有判断的机能。人們已逐步知道，数字型計算机可以进行（人类的以及机器的）語言翻譯，博奕（下棋），模仿其他的計算机活动等等，推而广之，人脑活动功能的一部分，可以在計算机上以小規模照样实現。数字型計算机，虽說是从簡单的計算机发展出来的；实质上它是超过单純計算机之上的某种东西，从而引起了很多人的兴趣。在这个意义上，本編专門叙述数字計算机。下編叙述数字計算机的設計基础，它本身也可看成是一个新的应用数学分支，即**自动机**問題的概論。

§2 什么是数字型計算机

目前，一般称作电子計算机的机器，从机能上讲，都是程序控制的自动計算机。这种計算机，从計算的机能上看，本质上是和台式手搖計算机相同的。

以普通的手搖式計算机为例，它有三个寄存器（register）：置数鍵盘部分（称为 **M 寄存器**）；記录手搖次数的部分（左边的記数盘，称为 **R 寄存器**）；得出答数的部分（右边的記数盘，称为 **A 寄存器**，亦称**累加器**）。計算按下列形式进行：

(M 的內容) × (R 的內容) + (A 的前存內容)

= (A 的新內容)。

（但是 R 的內容不是自动放上去的，R 仅仅在手操作时，記录轉动次数而已.）在称为**連乘式**的机器上，能将 A 的內容移到 M 中去，

所以象在

$$((a\times b+c)\times d+e)\times f \tag{2.1}$$

的計算中，中途就不必将答案写在紙上。然而計算下式时：

$$a\times b\times c+d\times e\times f, \tag{2.2}$$

必須先計算 $a\times b\times c$，并将結果写下，再計算 $d\times e\times f$，然后加上写下的数。同样的例子很多。

可見，計算(2.1)时，計算机中的三个寄存器是非常有效地工作着的，能将中間結果保存；而在計算(2.2)时，却少了一个必要的寄存器。

如果手搖計算机有更多的寄存器，以便数字出入寄存器之間則更好。但相当复杂的計算，难免要用手将中間結果写在紙上，这就容易产生錯誤。

这里可以看出，自动計算机的第一个条件是应具有大量存放中間結果的寄存器。这种寄存器的集合称为**存貯器**。存貯器中大量的寄存器，每一个都指定一个編号，称为寄存器的**地址**，将各个寄存器称作存貯单元。

在用机器进行計算时，如果缺少了存貯器，則一連串的計算所涉及的全部数据，随着計算順序的移动，每次都要取出，这些用手去完成将非常困难。**程序控制方式**就成为自然的发展結果了。这就是将原来用手操作的程序列成**代碼**(code)，再記录到紙带或别的什么东西上，利用它来控制机器，使机器自动地在存貯器的各个地址上放入或取出数据，以正确的順序进行計算。这种計算机就是**自动計算机**。

关于自动計算机的程序，早期的**电气机械式**(包括**继电器式**)的机器，皆以穿孔紙带进行控制，紙带通过紙带讀出机順序地讀出并控制操作(**紙带控制方式**)。在 ENIAC 中，使用了电子的运算装置，每秒钟能执行 5000 次加法，但紙带傳送速度限制了这种装置

的可能性的发揮。这就需要寻找更快速的控制方式。出现了利用插綫板上插綫的不同位置来改变內部电路情况，从而控制計算机使它正确执行的方式(**插綫板控制方式**)。这种方式虽然和电子的速度适合，但新的計算需要重新配綫，这是非常不方便的。

最好的方法是 von Neumann 提出的。他将程序也存入存貯器，机器依次讀出这些控制操作的程序，以实行控制(**內存程序方式或存貯程序方式**)。这是数字計算机前进中决定性的一步。由于这种进步，上述的(插綫板)不方便消除了；也开辟了計算机本身在执行程序过程中改变程序的可能性，从而使得計算机在应用方面获得无限丰富的变化。事实上，现在的电子計算机都是用这种方式控制的。

电子計算机具有

(i) **运算器** 具有与手搖計算机相同的机能。

(ii) **存貯器** 用于存貯中間結果和程序。

(iii) **控制器** 将程序的代碼譯碼，并发出各种指令至各部分，从而控制計算操作。

以上是計算机本体所必須具备的三个組成部分。除此以外，还需要某种机械在开始时将程序送入存貯器，这就是

(iv) **輸入設备** 从穿孔紙带或穿孔卡片上讀出信息，轉变为电信号，送入計算机內部。

此外，計算的最后答案，需要以我們能够辨别的形式給出，

(v) **輸出設备** 从計算机內部将电信号輸出，印刷数字，或以后再在紙带或卡片上穿孔。

以上五个部分是組成計算机的必要部分，图 2.1 表示各部分間信息和指令往来的关系，有关这些部分的实际构造将在第二編中讲述。因为使用計算机的人必須知道程序的編制方法，所以以下讲述理解程序构成的必要知識。

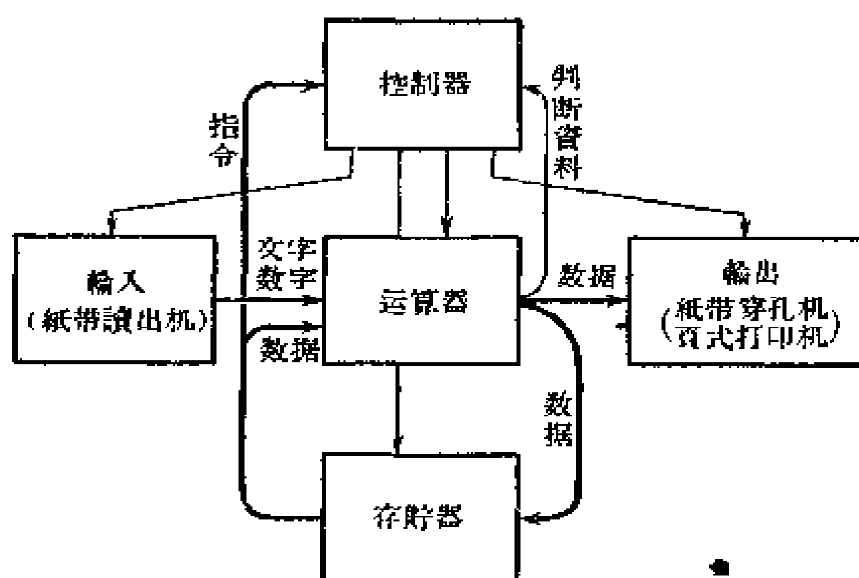


图2.1 存貯程序式計算机的結构

§3 計算机的程序(1)

今后主要考慮存貯程序式計算机。

使用計算机,首先要考慮把問題分解成具体操作的序列,其次将这个序列排成可以輸入計算机的程序;后者称为計算机的**程序設計**。

虽然我們可以按照計算順序作一般的研究,然而具体的計算步驟,却应根据不同的計算机的特性分别考慮。这是因为各个电子計算机所具有的运算操作种类并不一定相同。实际上,所有的电子計算机都具有加、减、乘三种运算能力,而除法設备却不一定都有。当然,由此在計算上造成一些困难,但也可由加、减、乘組合,以获得适当的完成除法的方法。有些計算机除了有除法設备外,还有开平方設备。此外,机器所取的数的范圍是有限的,太大的数不能进行計算,因而有自动校对小数点位数的机器(浮点式)。

另一个费事的問題是书写程序有不同的方式。因为,每个計算机各有自己的"語言",这种"語言",从单字到"文法"都有差异,

从而由一种語言翻譯到另一种語言是不簡单的。

然而，如果熟习了一种計算机的程序設計，那末其他計算机的程序設計也就不难了。現在以一个假想的計算机（我們的計算机），它具有基本的运算机能，来說明程序設計的特点。

計算，就是将一个个的基本运算反复执行，各个基本运算以一个**指令**（order, instruction）的形式給出。程序，就是指令的罗列。

一个指令由两部分組成，即表示**运算类型**（function, operation）的部分；指出存貯器中哪一个地址的数参加运算，以及答案存入哪个地址的**地址**部分。

数値或指令都存放在存貯器中，同样以地址标志它在存貯器中的位置。在一个地址中，普通存貯十进制十位左右，或二进制 40 位左右的代碼。在一个地址中这样貯存的信息，称为一个**字**。指令也当作一个字考虑。再者，当一个字的位数足够放置两条指令时，那末，一个地址就分成两半，放置两条指令。后者称为短字或半字。

我們的机器，使用正負 9 位十进制的数値范圍。負数以"10 的补数"表示，即 $-A$ 以 $10^{10}-A$ 表示。如

$$-135792468 \text{ 以 } 9864207532$$

写出。最高位（10^9）的数表示符号，以 0 表正数，9 表負数，称为符号位。符号位只能使用 9 或 0，如出現其他的数字，表明有了錯誤，称为**溢出**（但是在指令中可以出現其他数字）。

存貯器約有 1000 个字的容量，地址碼从 0～999 三位数字表示，运算类型以两位代碼表示，所以，一条指令是半字长的。

表 3.1 表示計算机的各条指令，其意义可由下面的例子了解。

今举一簡单的例子。

表 3.1

Acc—累加器 (m)—m 地址的代碼 [m]—m 地址的内容

[Acc]—Acc 的内容

类别	代码	内容
运算指令	11	清除 Acc* 的内容，[m]→Acc
	12	清除 Acc* 的内容，−[m]→Acc
	13	[Acc]+[m]→Acc
	14	[Acc]−[m]→Acc
	15	[Acc]×[m]→Acc*(1)
	16	[Acc*]÷[m]→Acc
	17	[Acc*]×10^{m}→Acc*(2)
	18	[Acc*]×10^{-m}→Acc*(3)
送存	40	[Acc]→(m)
	41	[Acc]4 舍 5 入后→(m)
輸入	21	(輸入寄存器的数)→Acc
輸出	50	将 Acc 前 m 位数向紙带上穿孔
控制指令	31(71)	向 m 号地址内的左(右)指令无条件轉移
	32(72)	当[Acc]<0 时，向 m 号地址的左(右)指令轉移
	33(73)	当[Acc]溢出时，向 m 号地址的左(右)指令轉移
	34(74)	当[Acc]≠0 时，向 m 号地址的左(右)指令轉移
	00	停机

(1) Acc* 是在 Acc 9 位寄存器后面再增加 9 位的长寄存器，可以容納 9 位数相乘所得的 18 位結果。

(2) 实际是向左移位，最高位的数字移入符号位，符号位的数字丢失。

(3) 实际是向右移位，Acc* 的最低位以下的数字丢失，而最高位 ≥5 时以 9 补充，≤4 时以 0 补充。

例 1 計算 $y=\dfrac{ax^2+bx+c}{dx+e}$，$x$ 从紙带順序輸入，各对应的 y 順序穿孔輸出，其他数值的位置如下：

$$[1]=a,\ [2]=b,\ [3]=c,\ [4]=d,\ [5]=e,$$

分别存貯在存貯器中。

解 按 §2 的說明，将分子写成 $(ax+b)x+c$ 的形式，中間結果每次都要用到，不必进入存貯器而可留在累加器中。然而在計算分子前后，須将分母的結果保存起来，同样，x 也須保存

起来。

表 3.2 是具体的程序。分母計算后的結果保存在[6]中。然后計算分子，再以除法将結果算出。除法的商数在累加器中給出。0 号地址和 6 号地址是暂时寄存数据用的，最后的指令将特别說明。

表　3.2

	指　令	指令内容	Acc 的内容及其他
100	L 21　000	IR→Acc	x
	R 40　000	[Acc]→(0)	$(0)=x$
101	L 15　004	[Acc]×[4]→Acc	dx
	R 13　005	[Acc]+[5]→Acc	$dx+e$
102	L 40　006	[Acc]→(6)	$[6]=dx+e$
	R 11　000	[0]→Acc	x
103	L 15　001	[Acc]×[1]→Acc	ax
	R 13　002	[Acc]+[2]→Acc	$ax+b$
104	L 15　000	[Acc]×[0]→Acc	$(ax+b)x$
	R 13　003	[Acc]+[3]→Acc	$(ax+b)x+c$
105	L 16　006	[Acc]÷[6]→Acc	y
	R 50　010	[Acc]→OR	将 y 穿孔
106	L 31　100	轉向 100 地址左指令	

最后的指令是**轉移指令**。它不是运算的指令，而是控制程序进行的指令。正常的程序进行时，是在一个地址中放置两条指令，首先执行左边的指令，然后执行右边的指令，以下依次进行。然而遇到轉移指令时，下一道指令不是依次执行，而依照这一指令地址部分所指示的地址取指令执行。在这一指令执行以后，仍依次执行。表 3.2 中最后的指令指出轉移到(100 L)号地址(即最初的指令)，也就是返回出发点的意思。

21 是**輸入指令**，它只是把輸入寄存器的内容送入累加器，因此与 11 指令有类似的性质(这个指令的地址部分和机器的操作是无关的)。輸入寄存器和存貯器同样有 9 位数字位和一个符号

位❶，用以寄存数据，数据是从紙带經紙带讀出机送来而存入的。在輸入指令到达时，一面送出輸入寄存器的内容，一面紙带机就自动启动，又将必要的数字位数装滿輸入寄存器。如果下一个輸入指令已經出現，而紙带还未讀完，显然这时輸入寄存器沒有装滿。这时計算机虽本身已准备就緒，但仍要等待輸入寄存器装滿之后，才执行輸入指令。这样，在紙带讀出的过程中，計算机具有**等待**的机能，这是輸入指令和簡单运算指令所不同的地方。

輸出指令50和送存指令40是类似的，是将累加器的内容移入輸出寄存器。当輸出寄存器装滿后，穿孔机自动启动，按指令地址碼所指定的位数将輸入的数字穿孔。穿孔不結束，下面的輸出指令不执行，直等到穿孔結束再执行。这一点和輸入指令是相似的。可以将紙带輸入另外的讀取机，也可以将輸出送至頁式印刷机，将結果印出。这时，数字的排列形式(layout, format)須由紙带上的特殊符号控制(如空白(space)，托架退回(carriage return)，換行(line feed)等)。計算机中必須能将这些符号在穿孔带上穿出，关于这点的說明从略。

以上程序在計算时，应注意数据和計算的結果不要超出計算机所容許的位数。

在乘法指令(15)运算时，二个9位以内的数相乘得18位以内的数，累加器应在低位附加9位，成18位的寄存器(Acc*)才能完全容納。但在执行40指令送存时，只考虑Acc，只将高位的9位存入存貯器。因此两个整数相乘时，Acc中的新内容是$[\mathrm{Acc}]\times[m]\times10^{-9}$。或者把所有的数都看成是小于1的小数，小数点在符号位的右面，而乘法只看成簡单的相乘。无論怎样处理，乘法的結果不会过大而造成溢出。

加减法指令(13, 14)需要小数点相同。而两个数相加的結果如超出計算机的取数范圍，則符号数字就变成0, 9以外的(1, 8等)数。

---

❶ 程序輸入时，为方便起見，实际上輸入寄存器作为将一个字的剩余部分收留的作用，这点将在§11中加以說明。

最容易发生溢出情况的是除法(16)及左移(shift)指令(17)，在除法中，当除数比被除数小时，結果商数就大于1，而超出了計算机的范圍。在上面的程序中，包含除法的計算，須使分母比分子大，两方面的标尺(scaling)取得不当时，就会溢出而造成停机。左移指令也是同样的，当累加器内数据很小时，左端很多位都是0(或9)，为了防止以后計算时损失有效数字的位数起見，就采用左移操作。这是左移指令的主要用途。如果累加器中的数不太小，那末左移过多，当然会形成溢出。

三地址方式及其他

以上所讲計算机的指令，原則上是一个指令中有一个地址，从存貯器中将这个数取出进行运算。但是，与这种单地址方式同样广泛使用的，有一个指令中包含三个地址的計算机。在这种計算机中，指令的形式是“[L]与[m]运算，結果放到(n)中去”。这种三地址方式，在程序中不必考虑累加器。例如，在我們的計算机中要用三条指令

11　　001

13　　002

40　　003

在三地址計算机中只用一条

1　　001　　002　　003

就够了。这条指令的长度增加对輸入是有利的；但在例1中，累加器被充分地利用了，40指令用得不多。但用三地址指令編的程序，其中指令的数目也并不减少很多。两者得失相半，現在都被广泛采用。

另外，还有在指令中指示下一道指令的地址的方式。相当于单地址方式的有二地址(即(1+1)-地址)方式，相当于三地址方式的有四地址((3+1)-地址)方式。这里，每一条指令都将轉移指令包括进去了。特别对于使用磁鼓一类的有等待时間的存貯器的計算机，为了减少等待时間，須将指令的排列自由变化，往往就采用这种指令方式。对此本文不深入討論。

§4　計算机的程序(2)——程序框图

例2　与例1相同，但 x 由 x_0 开始，按步长 Δ 增加，$x_0+\Delta$，$x_0+2\Delta$，…，$x_n=x_0+n\Delta$ 計算其数值，并将結果穿孔。且 $[10]=x_0$，$[11]=\Delta$，$[12]=x_n$。

解 在前例中，新的 x 值由 IR 輸入，当然每次按給出的 x 計算。現在，各 x 值要由計算机算出。

最初，輸入 x 的初始值 x_0 的指令先来，以后是与例 1 相同的計算指令。

最后，将 x 加 Δ，置于累加器中，并轉移到指令(100 R)(例1)，再进行新的 x 的計算。

随着 x 的逐漸增加，計算将不断继續下去，但到 x 等于 x_n 的时候，計算必須終止。因此每計算一周之后，即将 x 和 x_n 比較一下，当 x 等于 x_n 时就停止計算，这种判断机能是必須的。32,72 条件轉移指令的設立目的就是为此，即观察 $x-x_n$ 的值是 <0 还是 $\geqslant 0$. 若 <0 则計算出 $x+\Delta$ 并且仍如前法計算；若 $\geqslant 0$，則下一道指令执行**停机指令**，使机器停止操作。条件轉移指令附有某一条件(例如这里的[Acc]<0)，如这条件被滿足，則实行轉移，如这条件不滿足，則不实行轉移，而执行程序中的下一指令。

根据以上所述，可构成如表 4.1 的程序。

表 4.1

地址					
100	L	11	010	[10]→Acc	初始值 $x=x_0$ 輸入
	R	40	000	[Acc]→(0)	将 y 計算并穿孔
⋮	这中間的指令与例 1 中的(101～105)相同				
106	L	11	000	[0]→Acc	x 与 x_n 比較
	R	14	012	[Acc]−[12]→Acc	
107	L	32	108	[Acc]<0 时向 108 地址轉移	
	R	00	000	停机	$x\geqslant x_n$ 时停机
108	L	11	000	[0]→Acc	$x<x_n$ 时 $x+\Delta$ 代入 x
	R	13	011	[Acc]+[11]→Acc	
109	L	71	100	轉移至 100 地址	
若将各个 Δ 代以 $x_n+\Delta$ 存入[13]，則程序的最后可节約一条指令					
108	L	13	013	[Acc]+[13]→Acc	$(x-x_n)+(x_n+\Delta)=x+\Delta$
	R	71	100		

例 2 是**循环程序** (cyclic program) 的基本形式。可以分成五个部分。

(i) 初始程序　反复地进行程序前的准备手續，輸入全部常数及变参数的初始值等。

(ii) 主程序　执行实际运算的部分。

(iii) 分支　当一次操作完毕，进行判断并决定程序进行方向的部分。

(iv) 修改程序　在进入下一周期前，将必要的数值、地址进行修改的部分。

(v) 結束程序　包括获得必要的結果后，将它存入适当的单元，或将其印刷，并且把已修改的部分复原等等的最后手續。图 4.1 是以框图形式画出。好象田徑运动中赛跑的跑道，出发点与决胜点是两支分路，主跑道是(ii)(iv)．

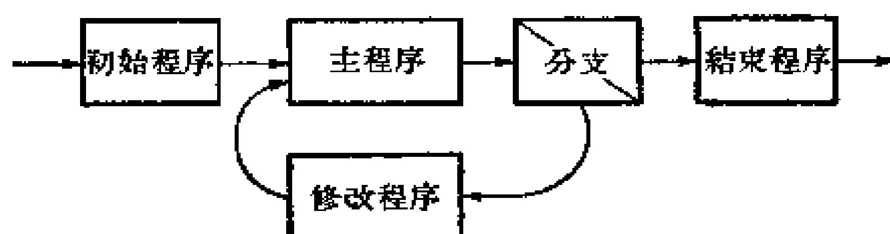


图 4.1　程序框图的标准形式之一

(iv)可以在(iii)的前面，也可在其后面，甚至在(iii)的前后都有。这种框图称为**程序框图**(flow diagram)，它能帮助我們考虑程序的設計。通常先写出程序框图，然后在每一个框中写入所操作的程序，以至全部程序都写入时为止。例 2 中可以写成

(i) 100 L

(ii) 100 R～105 R

(iii) 106 L～107 L

(iv) 108 L～109 L

(v) 107 R

实际的程序有时相当复杂，程序中可能包含二重至三重回路。但即使是这种情况，仍可用同样的方針对待。图 4.2 是两个行列式相乘的程序框图，具有三重回路，这个图中的修改程序是在分支前放人的。

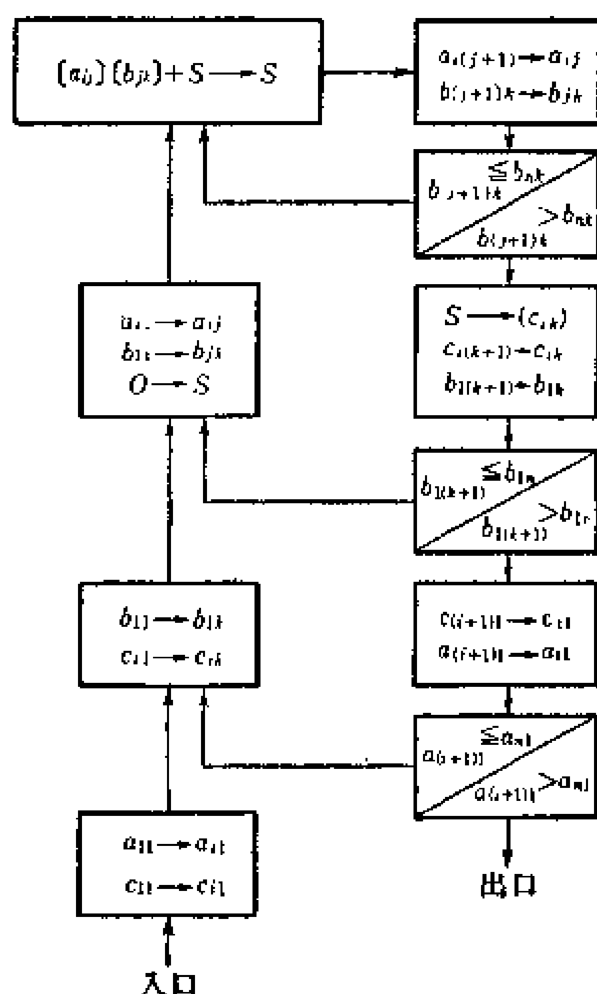


图 4.2 行列式相乘的程序框图

例 3 用逐次逼近法
$$y_n=\frac{1}{2}\left(y_{n-1}+\frac{x}{y_{n-1}}\right),$$
$y_n\to\sqrt{x}$ 求 $\sqrt{x}$，将其程序排出。但这个程序要能插入其他程序之間使用，在程序执行前 x 已在累加器中，在程序結束时也将 $\sqrt{x}$ 留在累加器中。0～9 的地址可供計算过程中使用。

解 上式的計算，即是所謂反复計算，相同的手續重复进行若干次即可，不必象前例中将参数变化。图 4.3 是它的程序框图，非常簡单，不必修改程序。每次反复完毕时，进行答案的近似程度的校驗。

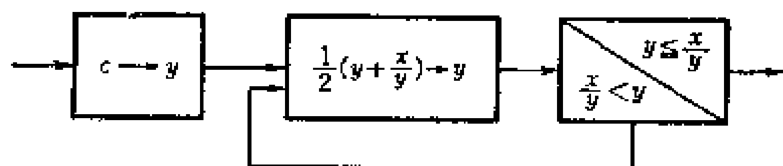


图 4.3 $y=\sqrt{x}$ 的程序

在这个逐次近似計算中，尽管以 y_0 等于 1 为試驗值（若是 $0<x<1$），也必然是收斂的，且每回的近似値都比眞值为大，精确度的判断以 $y_n\leq x/y_n$ 决定（y_n 比 x/y_n 小时，则当作 4 舍 5 入的誤差）。

考慮以上討論結果，可作出下列程序：

首先求出 $\frac{1}{2}\left(\frac{x}{y_n}-y_n\right)$，再加上 y_n，得 $\frac{1}{2}\left(y_n+\frac{x}{y_n}\right)$。前者便于精确度的判断，并可防止計算机溢出，是获得最大精确度的办法。

这个程序有两个常数 $c=1-10^{-9}$（初始值 y_0）和 $\frac{1}{2}$，在程序执行途中使用。一个常数在全部程序的其他部分并不使用，而仅属某部分的关系时，可作为程序的一部分将常数置入。

表 4.2

地址	指令		操作	说明
p	40	000	[Acc]→(0)	$x\to(0)$
	11	$(p+2)$	$[p+2]\to$Acc	将 $(1-10^{-9})$ 作为 y_0 存入
$p+1$	40	001	[Acc]→(1)	
	31	$(p+5)$	向 $(p+5)$ 的左指令轉移	
$p+2$	09999	99999	$1-10^{-9}$	第 0 次近似值
$p+3$	05000	00000	1/2	
$p+4$	13	001	[Acc]+[1]→Acc	以 $y_n+\frac{1}{2}\left(\frac{x}{y_n}-y_n\right)$ 代替 y_n 存入
	40	001	[Acc]→(1)	
$p+5$	11	000	[0]→Acc	$\frac{x}{y_n}\to$Acc
	16	001	[Acc]÷[1]→Acc	
$p+6$	14	001	[Acc]−[1]→Acc	$\frac{x}{y_n}-y_n$
	15	$(p+3)$	$[Acc]\times[p+3]\to$Acc	$\frac{1}{2}\left(\frac{x}{y_n}-y_n\right)$
$p+7$	32	$(p+4)$	若[Acc]<0，轉移到 $(p+4)$ 左指令	精确度判断，若不够则轉向 $(p+4)$ 左指令
	11	001	[1]→Acc	計算結束，$y_n\to$Acc

相对地址

在上面的程序中，有一些指令使用了象 $(p+3)$ 等的地址。这个 p 不是指存貯器中的位置，只是这样书写而已，实际上还必須以某一数碼代入。这样，在沒有知道这一段程序确实放在何处之前，就不可能把指令明确地写出。这样做带来很多不便，然而可想出很多方法来改进。

其中之一是**相对地址**方式，它采用附有特定符号的地址，不表示存貯器中的絕对地址，而以这个指令本身地址为基准，并以这基

准以后带多少个单元作为地址。例如将表4.2的$(p+1)$地址的右指令写成31*04的形式，*就是相对地址的記号，它的地址实际上是$(p+1)+04=p+5$地址。它的应用只限于具有这种設备的計算机。

另一种更一般的方法是在最初編程序时仍用适当的記法表示，而当指令輸入存貯器时，修改成实际的地址。例如，地址为$p+m$时(p是程序的最初指令所在位置)，在紙带上写着m^*即可(这是用适当的**輸入程序**使程序可以机械地改变并輸入存貯器)。例如，例3的p地址右指令$11(p+2)$，可写成11002^*．計算机遇到*就会在这地址碼加上p值。这是相对地址方法的一种。

以后在所討論的程序中，用某字代表地址时都用此方法表示。

§5 指令的修改

在反复程序中，不仅需要将数值規則地变更，而且数值的讀出位置，也就是說指令中的地址碼也需要規則地加以变化。在存貯程序式計算机中，指令本身可由累加器来处理，因此指令的修改是容易的。在非存貯程序式計算机中，指令的修改就要具有特殊的設备。

例4 計算多项式

$$y=\sum_{r=0}^{n}a_rx^r\equiv a_0+a_1x+a_2x^2+\cdots+a_nx^n,$$

a_0, a_1, $\cdots$, a_n存入$q\sim q+n$地址，且$[0]=x$，答数y放入(1)．

解 与例1同样，写成$((a_nx+a_{n-1})x+a_{n-2})x+\cdots$，从高次方开始計算，可使計算程序具有很簡单的形式。然而，这样的程序包含$2n$个指令，当n很大时，将占用很大的存貯量。于是，可利用計算的规律性，将同样的指令，組成反复形式的程序，使指令的地址修改成例如下面的形式：

常数存入

$$[998]=0000100000, \quad [999]=0000000000$$

表　5.1

地址	左指令	右指令	說明
0*	11　999	40　001	計算的初始值 [1]=0
1*	11　008*	40　002	將計数器(2)置数
2*	13　007*	40　004*	組成可变指令
3*	11　001	15　000	計算$[1]\times x+a_r\to[1]$
4*	(00　000	00　000)[1]	←此指令在执行 2*R 时輸入
5*	11　002	32　009*	結束的判断
6*	12　998	71　001*	計数器計数
7*	13　(q)	40　001	指令 4* 的原形
8*	00　(n)	00　000	計数器的初始值

(1) 指令中带(　)的部分在計算时將变化，它只表示初始状态。

这个程序以(2)号地址作为計数器，計数自 n 开始，每經过一次循环，卽減去 1. [4*]的左指令是可变指令，在每次循环中，由这个指令的原形[7*]加上計数器的內容而得到。(1)号地址保存中間結果，最后的答案也留在那里。

可变指令本身也可当作計数器利用，如表 5.2 所示，此程序每次將 1* 和 8* 比較，若 1* 小于 8* 就是計算結束。可变指令从最初放入的初始值开始，在每次循环的演算指令中間制成。第一次循环中的演算是无意义的，卽使不做也无妨碍。

从此例可見，在循环程序中，眞正的計算指令沒有地址修改和結束判断的指令多。当 n 很大时，若逐条写出程序，指令就要很多了。但是循环程序比直綫式程序所用的計算时間要长，在以計算

表 5.2

地址	指令	說明
0*	11 007* / 40 001*	放入 1* 右指令的初始値
1*	(00 000 / 00 000)	演算
2*	40 001 / 11 001*	修改地址
3*	14 998 / 40 001*	
4*	14 008* / 32 009*	結束的判断
5*	11 001 / 15 000	演算
6*	71 001* / 00 000	
7*	11 999 / 13 $(q+n)$	可变指令的初始値
8*	11 999 / 13 (q)	可变指令的最后値

速度为主的場合，只能使用直綫式的程序。

书写很长的直綫式程序，以及在紙带上穿孔都非常麻煩，且容易产生錯誤。使計算机自己按正确的規則作成指令，避免多用手穿孔，这种应用“編程序的程序”的方法，在存貯容量較大的計算机中，是比較重要的。

§6 記号地址

到現在为止，已經讲了几个簡单的例子，这是为了說明关于程序設計的要領。再作进一步深入时，思考的方法也是相同的。要用計算机解决問題有三个阶段。

（i）詳細分析問題的数学手續、次序，分解成方框，画出程序框图，

(ii) 将每一个方框中的計算分解,一直分解成(例如)象单地址計算机中的指令那样的单位操作。

(iii) 将单位操作的对象和結果的存貯位置指定,也就是将指令的地址指定,并且指令的輸入位置也指定,以完成程序的形式。

前两个阶段是本质的部分,要求正确掌握計算的手續,为了使刻板的計算机能"理解"所求解的問題,这是不可缺少的步驟。

在此阶段,使用 a, b, c 及 m, n, x, y 等記号代替演算内容,并以某些記号代替存貯地址,以代替眞正的指令写出代数式。这时看来很簡单明了,但实际是相当复杂而容易出錯的。以不同的記号书写的信息,假如在时間上不冲突,可以使用同一地址单元。因此,实际上需要多少地址单元是不太知道的,如果一开始就用了太多的地址,地址的节約又成了問題。

在指令中,轉移指令,修改指令等,常需指出某一指令的地址,因此他們决定于各指令在程序中的排列位置,如果排列次序未定,他們也定不下来。如果将所有指令的地址都已規定了,那末一旦发生錯誤或需插入指令时,有关指令的地址将全部变动,在更改时难免有所遺漏。

这是很不方便的,結果往往程序完成不了。这样就不得不采用一些語言代替直接編号,以适当的記号表示地址,将程序全部写出。

这时,某些指令的地址"被引用"于其他指令,在他們的左边注上适当的記号(引用符),同时作为地址使用了。

使用記号地址之后,改变程序的一部分,或增減指令,不会涉及編号的改变。这种形式非常便于推敲改錯。当最后沒有錯誤时,便可将地址分配(allocation)确定,并将全部編号写出。

最后一个阶段,完全是机械的任务。可用計算机帮助人去完成,錯誤既少速度又快。最近許多計算机都具有輸入变换的程序,

便于接受直接以記号地址穿孔的紙带。这些記号，由文字和数字組成，具有六个字以内的語言或数字即可（如IBM“SOAP”）❶。不可否认，用了記号地址以后，使得編制程序的工作“亲切而容易”了。

将例4（表5.2）的程序用記号地址写出。

a:	11	$(m0)$	40	(b)
b:	(	)		
	40	(y)	11	(b)
	14	998	40	(b)
	14	$(m1)$	32	(c)
	11	(y)	15	(x)
	71	(b)	00000	
$m0$:	11999		13	$(q)(n)$
$m1$:	11999		13	(q)
c:	續下面的程序			

这里，$x=0$，$y=1$ 可写作

$$0=x,\ y❷.$$

同样，若程序的存放地址从150号地址开始，则可写作

$$150=a:11(m0)\cdots.$$

将$(q)(n)$并列书写表示两个符号的数值相加。这些記号穿孔后，通过适当的程序，轉換成眞正的地址。

§7 子 程 序

需要用电子計算机解决的問題，大多包含着复杂的計算，其程序都要由几百条指令組成。直到現在，我們所举的例子，不过是那

❶ SOAP—Symbolic optimal assembly program（符号的最佳汇編程序）的縮写。最初用于IBM-650計算机，后发展有SOAP II. SOAP語言編制程序可以利用一組符号語言卡片，将符号地址程序由机器自动翻譯出直接代碼，并且可以“最佳化”，几乎和手編制的程序一样好。——譯者注

❷ 这里0,1皆指地址号碼。——譯者注

种程序的一部分。但大程序是由小程序(单位)构成的。具有一定机能的局部程序称作**子程序**(subroutine)。

例如开平方，exp，sin，tan，log，arc tan 等計算初等函数的程序，以及内插式，数值积分法等許多問題，可以一次作成子程序，保存起来，在其他問題中使用时将很方便。保存这些子程序，可以写在紙上，但如果保存在紙带上，并在需要时用紙带复制机机械地全部复制下来，且由机器直接操作，则更加方便而少錯誤。

使用計算机时，大部分劳动都花在編程序上。如果有一种能“拿来就合用”❶的子程序，則使用时将非常省力。

要使子程序能拿来就合用，須特别編制。子程序的組成具有两种性质。

（i）**开式子程序**

这种方式仅是从大程序中取出一部分。例如，計算

$$\int(1-x^2)^{-1/2}dx$$

的程序中，包含开平方的程序。則使用一个开平方的子程序，在程序的中途插入即可。

（ii）**閉式子程序**

有时在一个問題中，要一再使用一个子程序，在計算差值时便是如此。例如，計算$\sqrt{x}+\sqrt{1-x}$时便是这种情况。如果将相同的开平方的子程序存放两个，則对存貯容量是很浪費的。因此只放一个，必要时再使用一次即可。

計算机的控制，是使指令群得以执行。下面执行什么指令是由指令群决定的。不同程序进入开平方子程序的最后地址是不同的，必須将此地址記录，以便子程序結束时返回那个地址。这种手續称为“**接头**”(link)(或返回、联系)。采用接头这个方法的子程序

---

❶ 这是形象化的指“通用子程序”或“标准子程序”。——譯者注

称为闭式子程序。

在考虑"接头"的方法时，应能机械地"接头"以确保指令的执行有特别的意义。大多数場合使用下面的方法。

第一种是简单的方法，以子程序的最后轉移指令返回主程序。在組成主程序时，并不知道子程序的长度，仅在主程序中占用1.5~2条指令作接头工作。

由子程序直接向主程序轉移时，每次給予一个特定的地址，例如10号地址。这里置入将要轉移的**指示**(directory)地址。这是間接由主程序返回的方法。这里的接头工作是在10号地址內置入适当的轉移指令，而子程序的最后总是放置着31010及71010❶。这种方法的最大好处是不必变动子程序，便于使用**固定存貯装置**存貯子程序。所謂固定存貯装置，是指只有讀出而沒有写入的存貯器，如切断了写回路的磁鼓、二极管网絡等等，常用于存貯子程序及常数。

在我們假想的机器中，一个字內放置着两条指令，通常为了修改一条指令，往往伴随一条无用的伪指令(dummy instruction)。

作为实际的例子，在主程序中放置如下的两条指令❷：

$$11\ (m+1),\qquad 40\quad 010$$

其后跟着

$$m:\quad ——,\qquad 31\quad (p)$$

$$m+1:\quad 71(m+1)\quad ——$$

子程序是从 p 地址开始，在10号地址的左指令中已置入了 $71(m+1)$，即轉向 $(m+1)$ 地址的右指令。

若希望主程序中少用接头指令，則可作如下安排：

$$m:\quad 11\ (m)\qquad 31\quad (p)$$

---

❶ 即总是无条件轉向10左(右)地址。——譯者注

❷ 这就是"接头"指令。即取出$[m+1]\to[Acc]$，$[Acc]\to(10)$。——譯者注

先将它自身[m]置入累加器❶，然后轉移到 p 地址❷。左指令是将指令本身(m 的內容)置入累加器，而将其中左边的第 3~5 位的 m 記录下来，以便子程序接下去。而子程序的最初指令为

p: 13 996 40 010

在 996 号地址内

996: 20 001 00 000

于是当执行了 p 左指令时得$[m]+[996]=31(m+1)$, $31(p)$，再执行右指令后，就将上述轉移指令置入 10 号地址。从而可以在子程序結束时，引导至$(m+1)$地址的左指令。这里(10)的右指令是无意义的。

这种方法有利于使主程序簡单，但缺点是不能存貯必要的計算数据。

子程序使用后有贖回的作用。象上例中 10 号地址就有将子程序用后贖回的作用。将贖回子程序的接头置于别的場所，如 11 号地址，則重复使用子程序将不会相互抵触，而由公共返回地址决定。

另有一种子程序，在微分方程数值积分等处，作为补助子程序使用，用于計算方程中的函数。这里，預見贖回到那个地址将比較困难。此时，每一个問題的补助子程序都应当写出，看成是主程序的一部分，而将这些子程序在主程序之間变換是不会产生什么困难的(在补助子程序中也使用其他常用的子程序，与数值积分的子程序是不抵触的)。

§8 子程序的参数

由前述已知子程序的重要性。这种"成品"不論对誰使用，都应当适合各种各样的要求，应具有通用和标准的特点。如用于行列式乘法的标准子程序，应适用于不論几行几列的問題。行列的数值可以任意选擇，并置人适当的地址，以控制計算的进行。

❶ 等待修改。——譯者注

❷ 即轉移到子程序的开始地址。——譯者注

这些数值是为了使用标准子程序的目的而置入某些指令的地址中，是由主程序的指令置数的 然而，有时有很多的指令需要置入，上述操作将拉長了主程序。而标准子程序的作用，就是为了縮短主程序。这时，是将子程序的責任轉嫁給主程序而已。因此，可将必要的参数插入子程序之內，而使子程序本身来讀取这些参数。

这时亦有两种情况。

第一种，在大程序中，这个子程序将用到很多次，参数值是变化的。这时称为**程序参数**(program parameter)，它們属于主程序的一部分，通常放在轉入子程序的指令后面。

因此，这里的接头应使标准子程序了解程序参数在哪些地址，并在必要的地址放入指令，以便讀出参数时使用。

第二种，所有将要使用的参数，全部随程序进入子程序，称为**預置参数**(preset parameter).

預置参数的輸入，由特別的程序对輸入程序加以加工，这种特別的程序若一个一个地作也很麻煩，可以利用輸入程序的机能，将参数設計成某种記号。例如

13　005(3)

这条指令在穿孔带上，其地址指示出应存貯至 a_3+5 号地址，a_3 是参数，存放在地址 3 中。如此，参数 a_1, a_2, a_3, …，分别固定在地址 1, 2, 3, …中。利用輸入程序，将地址变换，称为地址形的参数輸入法，这是个主要的方法。

复杂的程序，往往包含很多的子程序，且其中的某个子程序要将其他子程序的指令修改，即要取出“某个子程序的某个字”，以这种地址表示法是很便利的。当然，眞正輸入計算机的，仍然是确定的地址，只在书写程序时，写各个子程序的地址，因为直接写出实际地址不方便。这时，将各子程序的进口位置(最初的地址)都取作参数，而在程序中按前述方法书写，即写成 13005(3). 其意义

是子程序中的第三个地址与第五个(从 0 地址开始的,即一般地址編号)地址的內容相加。

§9 解釋子程序

子程序的参数的一种用法,是指定被运算数的位置。試考察复数乘法的子程序。这个子程序的計算式是:

$$([0]+i[1])\times([r]+i[r+1])\rightarrow(0)+i(1),$$

亦即

$$[0]\times[r]-[1]\times[r+1]\rightarrow(0),$$
$$[0]\times[r+1]+[1]\times[r]\rightarrow(1);$$

其中 r 是参数 这个子程序示于表 9.1 中。

表 9.1 复数乘法的子程序

0*:	13015*	40002*	
1*:	13996	40010	接头
2*:	《11$(m+1)$	31003*》	讀参数
3*:	40006*	40010*	置入指令
4*:	13016*	40008*	
5*:	40012*	11000	$[0]\times[r]\rightarrow(2)$
6*:	《15(r)	40002》	
7*:	12001	31008*	$-[1]\times[r+1]+[2]\rightarrow(3)$
8*:	《15$(r+1)$	13002》	
9*:	40003	11001	$[1]\times[r]\rightarrow(2)$
10*:	《15(r)	40002》	
11*:	11000	31012*	$[0]\times[r+1]+[2]\rightarrow(1)$
12*:	《15$(r+1)$	13002》	
13*:	40001	11003	$[3]\rightarrow(0)$
14*:	40000	31010	
15*:	00001	00003	
16*:	00000	73000	

表中《 》中的"指令对"在計算时变化,这里列出它在計算过程中的形式,而在程序开始前放置 00000 00000。

使用此子程序的主程序,須在$(m+1)$地址上放

$m+1$: 15(r) 40002

即置放指定参数 r 的字(指令)。虽然在复数的加、减法子程序中，运算数的位置也是由参数指定的，但进一步将加、减、乘三种运算同用一个子程序給出是不可能的。

因为加、减、乘法的参数形式是各不相同的，所以还是各个問題用不同的子程序解决。計算时，主程序用共同的接头加以轉移。

例如，加减法的参数形式如下：

13 (r) 40000

及 14 (r) 40000

在适当的位置 a^* 放置比較的内容，

a^*: 15000 00000

子程序如下：

2*: ((11($m-1$) 31003*))

3*: 14 (a^*) 32 (b)*

4*: 13 (a^*) 31 (c)*

加减法子程序从 b^* 开始，乘法子程序从 c^* 开始。

更进一步看，每次使用象复数計算的子程序时，都将接头指令与轉移指令的参数轉回，这样做很笨。一种办法是一次进入子程序，不必轉回，不論哪一次計算，都当作普通的参数使用。此时只要将参数的讀出指令的地址每次加 1 即可。最后，由其他指令(如轉移指令)来区别参数的讀出，并从那个指令所在的地址轉回。

再进一步研究，所謂程序参数，从机能上分析，它可以理解为具备指定运算部分和地址部分的一个指令。但是，計算机的程序并不直接按照它的地址操作，却是把这个指令放入累加器，加以判别、修改，然后才显出它的实际效果。也就是說，計算机先对“指令”的各部分进行“解釋”，然后再执行。这与普通指令不同，它是非常間接地实行的。在这种意义上，将很多程序参数排列作成子程序，

称为解釋子程序或翻譯子程序。好比計算机不是用自己的語言，而是用其他的語言去閱讀，这就需要查辞典翻譯。而翻譯子程序就好比辞典和文法书。

这种方式与直接程序比較，費时較多。但从使用者的角度出发，只要写出簡单而必要的程序，是非常方便的。同样的方法可用于計算浮点及双倍长运算(18 位)。

利用解釋子程序进行浮点計算时，这种"指令"应采用完全自由的形式，不論是单地址或三地址，总之"指令"的种类或符号与計算机的固有指令无关，例如可含有开平方的"指令"。用相对地址来描写，則可产生各种各样的新机能。和反則地址范圍受到限制，机能縮小，然而有可能节約已經被指令挤得满满的存貯单元。

根据以上說明，实行解釋子程序和制造另外一台全新的計算机有相同的效果。然而那种新計算机的計划中，要在机器制造上投資巨費，那里以純机械指令实行解釋子程序。不用說，这种計划中，眞正的机器部分将耗費很多的时間和劳动，而解釋子程序則以較少費用，在机器性能即程序編制的容易程度以及計算速度等有关方面，得到很重要的資料❶。新的計算机常使用子程序，这些子程序都应通过眞正的机器加以檢查(因为复杂的子程序，即使由經驗丰富的人編制，也常易出錯，决不能自满)。

解釋子程序的方式，是最好地利用了电子計算机的特性，它使电子計算机具有"人工大脑"的机能的片段。后面将提到的 Turing 通用計算机的結构思想，在本质上与此是相同的，它意味着理解电子計算机的一种重要手法。

---

❶ 上述龐大的純机器指令的計算机計划并未实現，仅只在 1957 年被提出，是受了当时片面追求計算速度的影响。而后面的設想在最近几年的大型計算机中已实現了很多，如 ATLAS 就是以固定存貯器安置大量的子程序，而以控制器封鎖作用，形成广义指令，而实現了很广泛的复杂計算程序指令化的。——譯者注

§ 10 輸入程序(輔助程序)

計算的程序在最初是穿孔带或穿孔卡片的形式。將此程序利用机器的操作送入存貯器的手續，称为輸入。亦就是將紙帶(或卡片)上的穿孔順次讀出，并依次地送入存貯器的各单元。这可由机器本身构成这种能力，也可由短的程序去完成。例如

m:	21	000	40	(r)	从紙帶讀出并存入 r 地址。
$m+1$:	11	(m)	13	997	將上面的地址加 1.
$m+2$:	40	(m)	31	(m)	
997:	00	000	00	001	

全部輸入結束，輸入停止，开始执行程序的控制，由上述輸入程序的最后位置完成。在紙帶上从 1 号地址，2 号地址，……，直到 m 号地址的指令輸入，当 m 号地址的指令从紙帶讀出后，以 $(m+2)$ 号地址指令將 m 号指令置入，以后就轉向 m 号地址。于是新的指令就被执行。因此紙帶上 m 号指令对穿孔如下：

31 (p) 00 000

程序是从 p 号地址开始被执行的。

这 4 个字組成的程序本身如何輸入呢?

这里將使用一特殊的紙帶，称为**輔助程序** (bootstrap)．計算机本身只加一些附加設备，以使最初的几个字的程序能够輸入❶。

这个附加設备在"**启动按鍵**"按下时，將指令对寄存器先放置一条指令对

21 000 40 000

同时，用置 0 裝置將控制記数器(順序記数器) (control register)置于 0 状态。在执行上述指令对之后，即执行 0 地址的指令。

此时，紙帶的最初 3 个字如下：

21 000 40 001, 21 000 40 002, 31 000 00 000,

当启动开关按下，指令寄存器中的指令被执行时，紙帶的第一号指令对进入 0

❶ 由手控制开关板的电鍵輸入这条指令。——譯者注

号地址，指令的第二指令对进入 1 号地址，以后就按自己的規則进行。2 号地址的指令是使程序向 0 号地址轉移的。这时存貯器的內容是

0:	21 000　40 001	从紙带向 1 号地址放入指令
1:	(21 000　40 002)	执行所放入的指令
2:	31 000　00 000	

1 号地址內是置入紙带所指示的指令，以后立即执行❶。为了使前述的 4 个字輸入所指定的位置，紙带穿孔如下：

21000 40 (m)，　21000 40 (r)，　21000 40 ($m+1$)，　11 (m) 13997，
21000 40 ($m+2$)，40 (m) 31 (m)，21000　40997，　00000　00001，
31 (m) 00000，

共 9 个字。前述 4 个字应輸入 $m\sim(m+2)$，997 号地址，最后的 m 号地址是这个程序开始执行的进口。程序最初由 r 号地址开始輸入。紙带头上的这 12 个字就是所謂輔助程序。輸入指令时用五单位紙带讀孔机每次只讀一排孔，因此还另外需要**初始輸入装置**。

上述輔助程序稍加变化，将 0～2 号写在輸入程序內，則可縮短程序，且由下列

2100040001，　2100040002，　11(m)31000，　2100040(m)，
0000000001，　2100040000，　1300140001，　2100040(r)

8 个字构成。m 在程序最后第二个单元，輸入紙带的最后穿孔为 99999 99999 $(1000-m)$，下一个字的位置由 m 号轉回 0 号地址，再下面是 31(p)00000 穿孔，向 p 号地址轉移，則程序开始执行。

§11　輸入轉換程序

上述方法，仅能滿足指令及数碼向計算机輸入的最低要求，象前面讀到过的，在輸入时最好还要变更指令的地址。此外，在大多数实际的計算机中，运算都使用二进制，或者在輸入紙带上以等价于二进制的八进制，十六进制，三十二进制表示。实际上，这是不方便的記法，不习惯时容易产生錯誤。輸入紙带常以十进制表

❶ 这里 0 号地址是将拟存貯的地址置入 1 号，而 1 号就执行，将后一"指令对"輸入并存入指令地址，2 号是重复轉移。——譯者注

示数，必須用变換程序将輸入变换为二进制。这个稍稍复杂的**輸入轉換程序**(input conversion routine)，其本身由前述的簡单輸入程序从紙带輸入，之后，再輸入实际的程序。

輸入轉換程序的任务，包括加工指令，随时指定指令的輸入位置，以及在一条指令輸入后，控制程序的轉移等，从紙带讀出这些任务指示并加以正确执行。輸入轉换程序所用的紙带，往往写成如下的形式：

(m_1): 指令带 1，(m_2): 指令带 2，…，31 (p).

指令带 1 的内容从 m_1 开始，m_1，m_1+1，m_1+2，…輸入，指令带 2 的内容向 m_2 以下的地址輸入，最后，p 号地址是指示程序开始执行的位置。各指令对以逗点区分开来(若左指令是相对地址，則将星号記在中間)。最后的句点作为大結束。

例 4 (表 5.2)的程序从 150～158 号地址写在紙带上如下：

150: 11007*40001*, 4000111001*, 1499740001*, 14008*32009*, 1100115000, 71000*100000, 1199913000(5)(6), 1199913000(5),

将此子程序插入某主程序，該主程序已写至 149 号地址。紙带从这以后继續輸入，上面指出 150 号地址是必要的。这也就是指定相对地址的基点。

将 q, n 参数輸入 5, 6 号地址，当 $q=15$, $n=9$ 时，写成

5: 15, 9

上例是这个机器的輸入指令，开头亦可省略，而填写 0. 其差别在于作为特殊的輸入方法，在必要的情况下于 6* 号地址的指示中給出❶。

輸入指令

程序紙带上有各种符号，計算机的輸入指令(21)，除数字外，符号亦非讀

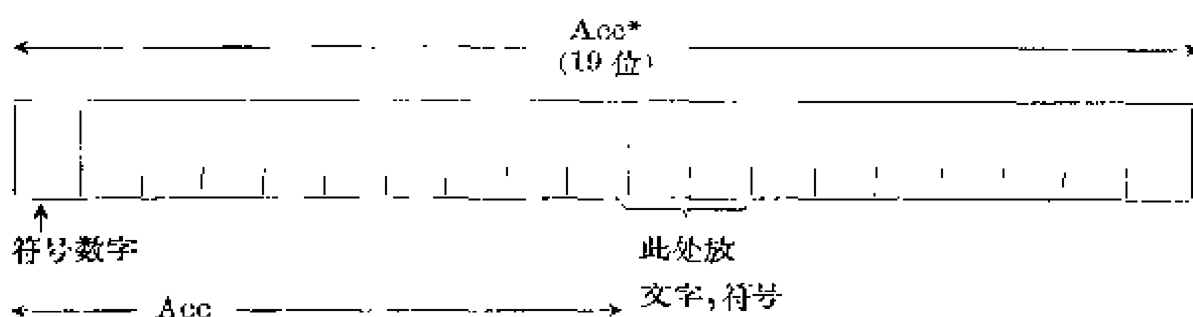
❶ 这样可以随时修改輸入的首地址，但今天一些机器的輸入指令已包括修改首地址的任务，所以已不是完全必要的了。——譯者注

不可。为了这个目的，"輸入寄存器"的10位再延长2位，各种符号就在这两位上編碼化。

12位的輸入寄存器的内容，移入 Acc* (連符号位共19位的延長累加器)的前12位。当紙带讀出时，由輸入寄存器的低位傳入。这时数字置于前10位，符号、文字置于最低的2位。

对于数字以外的文字、符号，有时需要和数字分开观察，或判别符号，皆由 Acc* 的后半段移位取出判别。

使用輸入指令，上述机能可由輸入轉換程序完成，这里不詳細說明。



輸出

計算机的計算結果，从累加器由**輸出指令**在紙带上穿孔或在紙上印刷。印刷工作是由頁式印刷器完成的。数字被适当地纵横排列，成为容易閱讀的形式。数与数之間及行与行之間須置以間隔，如每5行空一行。开头的非有效的0 (insignificant zero)不印刷(空白)，而＋、－、小数点等符号，都希望产生。这些工作如沒有文字印刷的指令是不行的。

例如59 $(\alpha\beta\gamma)$ $(\alpha\beta\gamma$ 表示数字) 这条指令是指"将对应 $\alpha\beta$ 数字的符号作 γ 次穿孔或印刷 $(\gamma=1, 2, \cdots, 9)$". 使用这条指令需要适当的輸出子程序。这样可排列成任意的表格。

在計算机中，有时也使用一次将一行数字全部印刷的行式印刷器。这时需要由計算机控制一个适当的緩冲寄存器 (buffer register)，将一行的内容全部排列好，由印刷指令执行印刷，印刷机上附有配綫板，以指定排列方式。

§12 小数点定位及其他

电子計算机的存贮器中的数，不过是只有一定位数的数的罗列而已。因此小数点的位置必須是固定的。在加减法时，小数点必須对位，乘除法时积与商的小数点的位数是不同的。

比較常用的原則是"小数点置于符号数字右边"。这时数字的范圍在 $-1<x<1$，这种方式的好处是乘法时不会溢出；除法时，当被除数的絕对值大于除数的絕对值时，无法进行除算，只得停机。

为了限制結果不得大于1，对于比1大的数或非常小的数如 $x=x_0\times 10^p$（十进制）或 $x=x_0\times 2^p$（二进制），必須利用比例因子，化成 $|x_0|<1$ 的形式，否則就无法进入机器。实质上，这是将小数点的位置改变。乘除法时，比例因子改变，而加减法时，对于比例因子的数，必須对位，使它們的比例因子相合，否則不能計算。这些問題在手搖計算机中同样存在，在电子計算机中只是計算的过程看不見而已，但仍是要注意的。

由于問題的性质不同，計算的結果有多大，并不都能看透，要預先把比例因子放在程序中是困难的。例如解算微分方程的情况，当包含奇点（含有∞点）时，解就急剧增加。当解增大时，要用比例因子将它变小，这就有必要組織自动校正比例因子的子程序。

为了避免这些麻煩，可用**浮点**(floating point)方式表示数值。在这种方式的机器中，数是以 $x_0\times 10^p$（或 $x_0\times 2^p$）（但是 $|x_0|<1$）的形式表示。将 (x_0, p) 两个数都存入存贮器，对这两个数进行計算，計算机就能自动地給出正确的小数点位置及解答。于是有

加减法 $(x_0, p)+(y_0, q)=\begin{cases}(x_0+y_0\times 10^{q-p}, p), & p\geqq q,\\ (x_0\times 10^{p-q}+y_0, q), & q\geqq p,\end{cases}$

乘法 $(x_0, p)\times(y_0, q)=(x_0y_0, p+q)$.

在加减法时，尾数(mantissa)部分若大于1，則应把小数点移前一位而在阶（指数部分）上自动加1. 按照需要可执行**规格化**的操作。有些机器是用特别的指令进行规格化操作的。所謂规格化就是把数表示成：

$$(x_0,\ p)\rightarrow(x_0',\ p'),$$

且 $x_0'=x_0\times 10^{\alpha}$,

$$10^{-1}\leqq x_0<1,\quad p'=p-\alpha.$$

表　12.1

浮点加法（用記号地址表示）

$[x]+[y]\rightarrow(z)$

a:	11 (x)	18002	将 x_0 与 p 分开
	40 $(x0)$	14 $(x0)$	
	17002	40 (p)	
	11 (y)	18002	y_0 与 q 分开
	40 $(y0)$	14 $(y0)$	
b:	17002	40 (q)	
	14 (p)	32 (c)	$q\geqq p$?
	14997	32 (d)	$q>p$?
	11 $(x0)$	18001	$\frac{x_0}{10}\rightarrow x_0$
	41 $(x0)$	11 (p)	
	13997	40 (p)	$p+1\rightarrow p$
	11 (q)	71 (b)	
c:	11 $(y0)$	18001	$\frac{y_0}{10}\rightarrow y_0$
	41 $(y0)$	11 (q)	
	13997	71 (b)	$q+1\rightarrow q$
d:	11 $(x0)$	13 $(y0)$	$x_0+y_0=z_0$
	17002	33 (e)	
f:	13 (p)	40 (z)	z_0, p 合成
	31 (g)	—	
e:	18003	41 (z)	$\frac{z_0}{10}\rightarrow z$,
	11 (z)	17002	
	13997	31 (f)	$p+1\rightarrow p$
g:	接后面的程序		

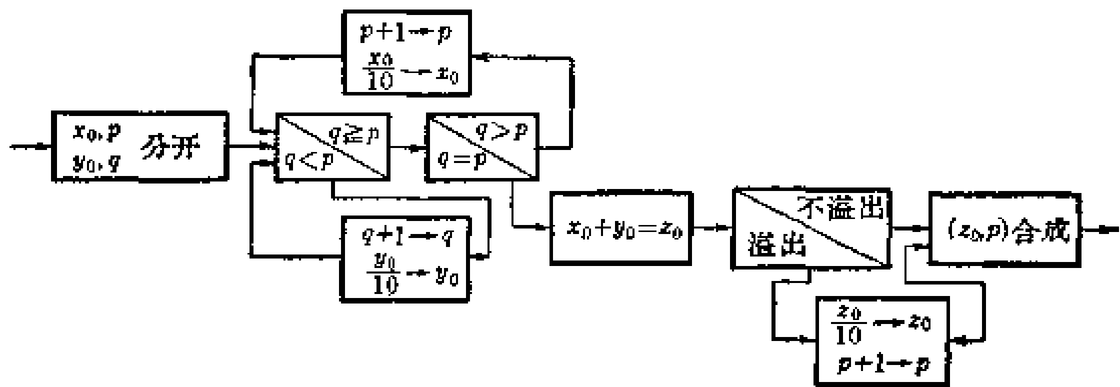


图 12.1 浮点加法的程序框图

次之，实行规格化操作，可以得到最大的精确度；但实际上对于精确度差的数使用规格化之后，反而得到表面上的高精确度。也有完全不实行规格化而經常只保持实际有效的数字位数的方法等等。这方面引起了不少爭論，可是还沒有一个完美的方法。

浮点运算需要相当复杂的設备，近来較大型的計算机多采用这种方式，中小型的机器却实际上不大使用。如果机器本身是定点的，也可以利用程序进行浮点运算（如表 12.1，图 12.1）．特别是利用象 §9 中的解釋子程序的方式，就可以将程序写成普通的型式，而机器就能进行象浮点方式的程序，这是使用浮点运算的解釋子程序所致。使用浮点方式使工作非常容易，且和使用浮点机器相比，一点也沒有差别。缺点是运算速度非常低（要慢数倍以至数十倍），而且編程序的容易程度比較差。某些問題，往往先在某些代表性的地方以浮点子程序計算一下，以便掌握解的大致傾向，了解比例因子，然后作定点的程序，对整个题目进行細算。实际的方法就是如此。对于具有浮点設备的机器，可以說，也需如此，浮点設备的机器設备多，而浮点計算，特别是加减法，不如定点快。在理論上浮点方式可以解决任何問題，然而有些可以用定点方式解决的計算問題仍用定点計算，效率比較高。因此对于具有浮点装置的机器，一般都有作定点和浮点两种运算的指令。

高精度的計算

計算机的固有字長(位数)(我們的机器是十进制9位)有时达不到精确度的要求。象級数展开,正負項相消以致低位不够保留差值等是典型的情况。这时可使用子程序,以机器原来2倍或3倍的位数进行計算。一般将数据分为高位和低位两部分(也可有更多部分),分別計算,然后合起来。手摇計算机也是同样作法。(除法的分开計算比較麻煩。)为此,利用解釋子程序就很方便。

§13 程序的檢查

使用計算机的計算都非常复杂,只要有一个錯誤,就无法获得正确的答案,所以檢查是重要的。一般檢查包括机器的檢查和程序的檢查两种。关于机器的檢查这里不談。

将程序通入机器,使用穿孔机器一次就得出正确的答案是不大可能的,必須考虑有二、三个錯誤。所以要定出发現錯誤的方法。

按电鈕使机器一步一步地进行的方法不很有效。这种方法化时多,而"計算机的时間"是很宝貴的。这里可使用特別的**檢查子程序**通入机器。这种子程序的种类很多,但大体分为两类。

1) **解剖程序**(post-mortem program) 当計算进行不佳而自然停机或因故中途停机时,就将特制的解剖紙带插入机器,解剖程序应送入空白的存貯单元后再启动。这个解剖程序将被檢查的程序所使用的計算单元的内容印刷出来,逐点判定錯誤所在。有种解剖程序将被檢查的程序带通入,并比較那个程序的最初状态和現在的状态,把变化的部分印刷出来,这样可以节約印刷的时間。亦还有将特定位置中的指令和地址全部印刷出来的。

2) **操作試驗子程序**❶ 这是一种解釋子程序,在执行程序的一条条指令时,附带执行与指令有关的适当信息印刷出来。印刷

❶ 从内容看,是属于"追踪檢查程序"。——譯者注

的信息有下列几种:

(i) 将每次执行指令的种类全部罗列印刷;

(ii) 仅将轉移指令及有关地址印刷;

(iii) 只在程序中某一部分到来时将这部分如何变化而执行的情况印出。因此把叫做封鎖指令的轉移指令插入原来主程序之中。只有这部分使用解釋子程序执行,所以并不比平常的操作慢多少;

(iv) 将累加器的内容在送入存貯器时印刷出来。

这类程序更接近邏輯的程序,編制时要更熟练和細致,它是計算机有效活动的主要子程序。

§14 展　望

在本文中,以一部假設的計算机为例,展示了程序編制的过程。本书中的計算机并不存在,也不是理想的計算机,甚至是一部非常不方便的、落后的計算机。对于現代計算机中所存在的不方便之处,并未能有所表現。

实际上,各个計算机的指令系統是不同的,亦即使用不同的"語言"。有些以少数指令产生很多的操作;相反,也有以极少数的指令,依靠巧妙的組合。比較优劣,很难用某种絕对标准衡量,因为随着使用目的、机器速度、存貯容量等标准的改变,結論也就不同。

这里只对今后电子計算机的傾向性問題,特别是"語言"方面作一些大概的介紹。但仅限于科学技术計算用的机器,而不談办公用(事务用)計算机。

1) **二进制与十进制**　我們的計算机是以十进制表示数和指令的,現代的計算机大都使用二进制作内部語言。由于十进制的每一位数是以二进制 4 位(bit)的数表示的,它的一位相当于二进

制的3.3位($=\log_2 10$),且运算器用二进制时能簡单而迅速。因此决定了大型、高速、大容量的計算机都采用二进制作内部語言。

二进制的缺点是进入机器的数值是以十进制表示的，答案印刷又要用十进制表示，这就很麻煩。但是这可用专門的子程序进行。程序編制者必需担負起这一任务。今后的趋势是"机器内部語言是十进制还是二进制与程序編制者无关",而是設計者考慮的問題。

在指令方面，十进制与二进制最明显的区别是移位指令。但浮点表示的数无須移位。这就是說，在数值輸入計算机的时候恐以十进制較好;除此之外,就不必区别是十进制还是二进制。答案的印刷希望用十进制。

在計算机中，除数字計算外，尙有将数与文字、符号等加以区别而选擇操作,以及相同的数或文字查出等邏輯的操作,此外还有修改地址的指令。这一切在浮点情况下是困难的，有必要使用定点方式。另外，尙需能将存貯內容的"每一位"都进行邏輯操作的指令(后述)。

2) **指令的修改**　在本文里，指令地址的修改，是将指令引入累加器进行。此时,对于指令对是不方便的,且对于存貯容量亦不經济。根据这些經驗得出下列几点改进办法:

(i) 設計时使能用半字长分别讀出写入(EDSAC, IBM-701).

(ii) 采取仅对左或右指令的地址部分在累加器中的相应位置作变换(ILLIAC 等)。

还有一些不方便，如循环程序中,每次修改时，都要使用累加器，这时計算的眞正数值必須暂避一下，于是常常要进出存貯器。因此

(iii) 設計一个小累加器(**B-寄存器**),以修改指令的地址。这

样，累加器只作加减法运算装置。修改地址在B-寄存器进行，并且B-寄存器的内容正負（零或非零）还可作为分支条件，設以条件轉移指令。

进一步考虑可以用下述方法：

(iv) 指令的执行，由指令所示地址加上B-寄存器地址，得实际执行地址。

因为也可能有不需修改地址的指令，所以在指令内有一B位(B-digit)来表示是否需用B-寄存器来修改地址。B-寄存器可以有几个❶，这时每条指令都指定了选擇对象（此方案为 Manchester 大学 Ferranti 提出）。

这种方法，当B-寄存器的数改变时，指令就改变，所以存儲器中所存的指令不必修改。这时可少用"存贮"指令。另外，每条指令都需将地址进行加法运算，因此指令的执行时間多少要长一点。

但是，一般說，使用了B-寄存器"自动修改地址"的机能后，簡单的循环程序显然将非常簡单，从而速度增加。且利用上述条件轉移的机能計算循环次数，可使程序从最后一圈自动脫出。

3) **浮点方式** 早期的計算机都是定点方式，而以子程序进行浮点計算。现今比較大型的計算机都傾向于机器本身裝置浮点設备。編制浮点程序不考虑比例因子，对于普通計算一般不常会溢出，仅仅考虑有多少有效位数，而定点方式就要花費較多的注意力于比例因子。

浮点計算的缺点，特别是加减法比定点方式时間长。然而对于規格化是随意执行的机器，在編程序时使尾数部分不大于1，而且不执行規格化，則实际上作定点操作（指数不变）。这时两方面

---

❶ 目前大型的近代化計算机，如 ATLAS，使用256个B-寄存器，称为变址寄存器。——譯者注

的优点都可取得。

4) **輸入輸出**　在我們的机器中，輸入指令由电傳打字紙带上的数和非数符号讀出，寄存于輸入寄存器中。在一般的十进制計算机中，从輸入紙带送入存貯器的操作很簡单，作为輸入紙带的讀出机，一般用高速光电讀出机。二进制的計算速度极高，光电讀出机的速度(每秒 200～400 字)还是不够，希望有一种十分快速的輸入存貯器的設备。

与穿孔带、卡片同样重要的輸入輸出装置有**磁带**，它比穿孔带的速度高。存貯器的內容可以直接記录到磁带上，并可逕向存貯器傳送，可作为比較低速的**2級存貯装置** (second-ary store)。常用子程序等記录在磁带上，必要时就讀出使用(二进制机器，直接記录二进碼)，但由于要反映相对地址或参数，因此需要有适当的方法。

輸出的方式，以穿孔带来說，只有 60 字/秒，比輸入还慢。可是电傳印刷机(电傳打字机)更慢(約 10 字/秒)。这时常將結果先在紙带上穿孔，借助发报机使电傳机印刷，利用輸出时間的不同而均匀分布，使印刷机发揮全部能力。然而在科学上計算問題所得的結果，其数值往往比較小，电傳打字机已足够，特别是后者的計算的結果，須保存在紙带或磁带上，这时可以用二进制(或十六进制，三十二进制等)直接記录。

5) **邏輯机能**　电子計算机的重要应用之一是**組合問題** (combinatorical problem)，例如，寻找"誤差校正碼" (error-correcting code)的实驗計划法等問題的应用。在这里，数的加減乘除操作不如邏輯操作重要。邏輯操作有 $\vee$，&，+(mod 2)等(参看第二編)。这些操作中，是將 m 号地址的数与累加器的数按位进行数字(0，1)操作，結果在累加器中保存，这是重要的指令。科学計算用的計算机中这一类指令較少(& 操作較多)。还有以 m 单元的数

与累加器中的数相等与否組成的件条轉移等，都是**翻譯**（人类語言的翻譯，計算机"語言"的翻譯）等重要任务所使用的指令。

另有与上述相关的机能，称为**探索指令**。即是"从存貯器的第 m 号地址开始，探出与累加器內容相同（或半字相同）的存貯单元"。这是个判断如上述相等不相等的指令，在存貯器中依次自动走查，寻到含有那个字的地址后，将此地址留在适当的位置（如B-寄存器）。

今后电子計算机的設計，将特别在这些邏輯机能方面以尽可能高的速度进行。邏輯性程序是最花費时間的程序之一，人們总希望它有最高的速度。

第二編　邏輯綫路和自动机理論

§15 序　言

在第一編中，把計算机看作是一架优良的机器，好象一个非常忠实而且一絲不苟的計算員，只要用某种特定的語言向他交待問題，他就能正确理解而給出解答。在本編中，将要进一步研究这机器的内部是什么东西？也許讀者覺得本丛书是应用数学的讲座，对于这些由电子管构成的复杂机器的内部构造，沒有必要去考慮，只要电气工程师知道就够了。其实不然，計算机的設計問題和計算机的程序編制問題一样，也是数学工作者的研究課題。

用几千个电子管构成的电子計算机，并非只是在大底板上将电子管、电阻器和电容器等随便接綫装配成的。現代的电子計算机都是以一个个具有一定机能的**机能单位**(functional unit)为基础，并以若干种这样的机能单位按照各种不同方式組合而成的复杂有机体。各种机能单位的設計当然是电子工程师的任务，至于如何将这些机能单位組合成計算机却仍然是数学上的問題。尤其是在采用“**变参元件**(parametron)”的計算机上就只有一种机能单位，因此把它們加以不同組合而設計計算机时就不必去考慮电方面的問題了。在这种意义上，把电子計算机作为数学对象而加以研究，当然要把它的动作加以某种程度的理想化，也即抽象化。Turing 計算机就是这样抽象化的計算机模型之一，将在后文叙述，这类机器的更一般化的概念即是**时序自动机**(sequential automaton)．实际制成的时序自动机是各种各样的，其中有自动售貨机、保險箱鎖等简单东西，也有自动电話交換机、鉄道信号轉轍

机的連动装置等龐大系統。自动机理論就是研究如何把这些装置加以数学抽象化，如何从簡单的自动机組合成更复杂的自动机，以及如何描述自动机的特征等問題。在本編中叙述与計算机有关的自动机理論基础及其在具体問題中的应用，特别照顾到計算机設計方面的应用。

第1章 邏輯代数

§16 邏輯函数和邏輯式

电子計算机，継电器計算机，今天的一切自动計算机，几乎都使用**二值的**电流或电压信号。計算机的各組成部分間以电綫連接，使数或指令的信息得以傳送。信息由各条导綫及各个瞬間的两个可能值表示，例如，电流是 I 安培还是 0，或者是 $+I$ 安培还是 $-I$ 安培。由于利用二值的区别来傳送信息，所以中間电流值是不采用的。而且，一对导綫在一个瞬間只能傳送一个二进制信息单位(bit)，假如十进制 10 位的数傳送时，就需要几对导綫，或者在一对导綫上一个接着一个的以电流傳送，用它們的組合来表达必要的信息。在計算机中使用二値信号，执行必要的四則运算，指令符号譯碼，控制信息通路开閉等机能都是**邏輯操作**(logical operations)，其数学表示为**邏輯式** (logical expression). 表达这些有关問題的数学工具是**邏輯代数** (algebra of propositions) 或**布尔代数** (Boolean algebra)❶. 这也是一般研究自动机理論的基础。

邏輯代数只处理两个"数"，通常写成"0"和"1". 但最好把它們看成和普通的数字0，1毫无关系，仅当作两个記号使用，即使不写 0，1，而写成"+"和"−"，"上"和"下"也都一样。

邏輯代数和普通代数一样，也用 A, B, C, X 等文字作为"变数"。文字之間的种种关系，是以式子表示的，即是在一般場合下用 0 或 1 代入后結合在一起所表示出来的。在这里，$X_1, X_2, \cdots, X_n$

❶ 表达邏輯关系的数学工具，不只是邏輯代数，还有很多其他数学方法，如遞归函数，拓扑学等。——譯者注

是**独立变数**，定义这些变数的函数 $Y=f(X_1, X_2, \cdots, X_n)$ 是**邏輯函数**(propositional function)。普通实变函数只要变数的可能值及函数的对应值唯一地确定，則函数卽完全定义；邏輯函数的 n 个(有限)独立变数以 0 或 1 代入，得 2^n 种組合值，一一对应地确定，則此函数亦完全定义。两个函数 $f(X_1, X_2, \cdots, X_n)$ 和 $g(X_1, X_2, \cdots, X_n)$，若 $X_1, X_2, \cdots, X_n$ 以 0 及 1 的一切可能組合代入常相等(两方都成为 0，或为 1)，則称 f 与 g **恒等**，写作

$$f\equiv g.$$

这两个函数，卽使仅有一种变数組合值代入而互异，則就不恒等，卽 f 与 g 是不同的函数。

因此，具体地定义邏輯函数的一般方法，是用对应表来表示的。这种 2^n 函数值的表称为**眞值表**(truth table)。表 16.1 是两个三变数的邏輯函数的例子，为了簡单起見，表上变数值对两个函数共用。这是二进制加法中必需的函数。

实数或自然数具体地表示物体的长度、电流的强度等物理量，或橘子的数目等；而邏輯变数則表示如明天是不是星期日，及格还是不及格，东京市区还是市区以外的地方等等两者之中是哪一个的**命題**，也卽是对某一問題的"是"还是"否"的答案，或表示一个命題是**眞**还是**假**。

表 16.1 眞值表的例子

X_1	X_2	X_3	Y_1	Y_2
0	0	0	0	0
1	0	0	1	0
0	1	0	1	0
1	1	0	0	1
0	0	1	1	0
1	0	1	0	1
0	1	1	0	1
1	1	1	1	1

邏輯函数的命題是眞是假，由其他的命題决定。例如，"他是我的弟兄"(X_1)，"他比我年紀大"(X_2)，"他是男的"(X_3)，这三个命題可組成邏輯函数(Y)"他是我的哥哥"。这里 Y 是 X_1, X_2, X_3 的邏輯函数。仅当 X_1,

X_2, X_3 全部为眞时，Y 才是眞的，此外都是假的。同样，“她是我的妹妹”(Y')这个命題也是 X_1, X_2, X_3 的邏輯函数，只有当 X_1 为眞，X_2 为假，X_3 也是假的时候，Y' 才是眞的。但“他是我的大哥”这个命題則不是这些变数的函数，因为仅由上述三个命題的眞假不能决定这个命題的眞假。

一般的邏輯变数 X_1, X_2 等，作为命題考慮时，規定当命題为眞时取值 1，当命題为假时取値 0．另外，由变数 X_1, X_2, …等构成函数 $Y=f(X_1, X_2, \cdots)$ 的手續称为**邏輯操作**。

将邏輯函数本身看成是一个邏輯变数时，可构成其他的邏輯函数。例如

$$Z=g(Y_1, Y_2, \cdots)$$
$$=g(f_1(X_1, X_2, \cdots),\ \ f_2(X_1, X_2, \cdots), \cdots),$$

式中作为 g 的独立变数的 f_1, f_2, …之中的独立变数，可以互相独立，也可包含共通部分。这样构成“函数的函数”的过程称为**邏輯操作的組合**。

与之相反，一个邏輯操作可分解为較簡单的邏輯操作的組合，而这个組合得到的邏輯函数与原来的相等。实际上，将任意的邏輯函数分解，最后可归結为 2～3 种最簡单的邏輯函数。取哪些作为**基本邏輯操作**，这在一定程度上是自由的，最通常的取下述三种。

表　16.2

A	B	$A\cdot B$	$A\vee B$
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

〔基本邏輯操作〕

I “**与**”(and)　以“A 与 B”, $A \& B$, $A\wedge B$, $A\cdot B$, AB 等表示二变数的函数。表 16.2 第三列是以眞值表所作的定义，只有当 A 为 1 “与” B 为 1 时才等于 1．若将 0, 1 解釋为“数字 0, 1”，則和普通乘积一致，称为**邏輯积**，可以用相同的記号书写。

II “**或**”(or)　以“A 或 B”, A or B, $A\vee B$, $A+B$ 等記述。表

16.2 第四列是以眞値表所作的定义。只要 A"或" B 中不管哪个是 1 它就等于 1，称为**邏輯和**，与算术和不同，以 $\vee$ 作为記号。

III "**非**"(not) 它是一个变数的函数，"非A"(not A)以$\sim A$，$\bar{A}$, A', A^{-1} 等为記号，本书采用記号 A'. 其眞値表示于表 16.3. 它又称为**否定**。

表 16.3

A	A'
0	1
1	0

将这些基本操作象代数运算符号一样处理，可写出組合的表达式。例如 $(A\cdot B\vee C)'\vee D$. 若規定"邏輯积"符号要比"邏輯和"符号优先进行，則括号可节省。

邏輯和，邏輯积与代数一样，对**交换律**、**結合律**成立。这由眞值表可明显看出。即

$$[1a] \qquad B\vee A\equiv A\vee B,$$

$$[2a] \qquad (A\vee B)\vee C\equiv A\vee(B\vee C),$$

$$[1b] \qquad B\cdot A\equiv A\cdot B,$$

$$[2b] \qquad (A\cdot B)\cdot C\equiv A\cdot(B\cdot C).$$

交换律成立时，象 $f_1(A, B)=A\vee B$, $f_2(A, B)=A\cdot B$ 等邏輯函数可认为是**对称的函数**。当結合律成立时，可略去括号，允許写成 $A\vee B\vee C$, $A\cdot B\cdot C$.

分配律也成立。

$$[3a] \qquad A\cdot C\vee B\cdot C\equiv(A\vee B)\cdot C,$$

$$[3b] \qquad (A\vee C)\cdot(B\vee C)\equiv A\cdot B\vee C.$$

这里，[3b]是新的关系式，用眞值表来驗証可知是正确的。至今所引出的定理都是 a, b 成对地出現的，操作 &, $\vee$ 也是成对地出現的，这是所謂**对偶定理**的一般性质。与普通分配律[3a]平行的是"加法分配律"[3b].

下面給出几个必要的定理：

$$[4a] \qquad A\vee A\equiv A,$$

[4b] $A\cdot A\equiv A$,

[5a] $A\vee A\cdot B\equiv A$,

[5b] $A\cdot(A\vee B)\equiv A$.

同样它們也表示为对偶的性质。

下面列举几个含有否定的定理：

[6] $(A')'\equiv A$,

重复地否定将回到原来的元素(否定之否定成为肯定)。若 $A'=B$ 則必然有 $B'=A$. 总之,否定的操作是相互的。

[7a] $(A\vee B)'\equiv A'\cdot B'$,

[7b] $(AB)'\equiv A'\vee B'$.

一个式子被否定,等于其中的变数否定并将 &, ∨ 操作交换。一般的形式可表述在下面的定理中。

定理 16.1 (对偶定理) 由变数 $X_1, X_2, \cdots, X_n$ 及 &, ∨ 組成的任意邏輯函数 $Y=f(X_1, X_2, \cdots, X_n)$; 将其中的 ∨ 全部改变成 &, & 改变成 ∨ (但括号的替換以原来操作順序不变为准); 得函数 $f^*(X_1, X_2, \cdots, X_n)$. 于是

$$Y'=f^*(X_1', X_2', \cdots, X_n'),$$

即下式成立:

$$[f(X_1, X_2, \cdots, X_n, \&, \vee)]'=f(X_1', X_2', \cdots, X_n', \vee, \&).$$

这个定理的証明,可由[7a],[7b]从括号外側开始依次应用本定理变換即可。严格的証明需应用数学归納法。

依据本定理,对一个邏輯等式两边进行否定,然后应用本定理,把所有变数以其否定置换,得到前面式子中 & 和 ∨ 已替换的式子。这样就証明了一个新的邏輯等式(即前式的对偶式)。由此也就可以了解为什么以前各公式都是成对出現的。

0 和 1 对偶的法則

以前把 0 和 1 作为邏輯变数所取的值。然而同一个記号也可

以用来表示**常数**，即永取恒値的邏輯"变数"。以 1 为"恒眞的命題"，0 为"恒假的命題"。下面的定理成立：

[8a] $A \vee A' = 1$,

[8b] $A \cdot A' = 0$,

[9a] $A \vee 0 = A$,

[9b] $A \cdot 1 = A$,

[10a] $A \vee 1 = 1$,

[10b] $A \cdot 0 = 0$.

使用以上各关系式，可对給定的邏輯式加以种种变形。例如，普通以邏輯积的形式展开的式子，使用[3a]可将任意邏輯式中的邏輯积操作代以邏輯和操作，得积之和的形式，即所謂加法标准形。当然，若有否定的記号就有必要用[7a]，[7b]把否定記号化到括号内去。例如

$$(A \vee B)\cdot(B \vee C \vee D) = A\cdot(B \vee C \vee D) \vee B\cdot(B \vee C \vee D)$$
$$= A\cdot B \vee A\cdot C \vee A\cdot D \vee B \vee B\cdot C \vee B\cdot D,$$

这个去掉括号的"积之和"形式称为**加法标准形**。一个式子写成加法标准形的方法不只一种(即使不考虑和的次序和各項中积的次序)。

蘊含(implication)

两个邏輯函数間的关系，如若 Y_1 为眞，則 Y_2 必然为眞，而其逆不一定成立时，写作 $Y_1 \subset Y_2$，或 $Y_2 \supset Y_1$。若 $Y_1 \supset Y_2$，而且 $Y_2 \supset Y_1$，則与 $Y_1 = Y_2$ 等价。于是，可設想这和不等式記号 $\leqq$ 相类似，且含有"若 Y_1 則 Y_2"的意思。故下式（**三段論**定理）成立：

$$A \subset B \text{ 且 } B \subset C \text{ 則 } A \subset C$$

又 $A \subset B$ 与下列各式是相同的：

$A\cdot B' = 0$ （A 是眞而同时 B 又不是眞的事情不成立）

或者

$$B\vee A'\equiv 1 \qquad (\text{要么 } B \text{ 是眞,否則 } A \text{ 也不是眞})。$$

另外

"若 $A\subset B$, 則 $A\vee B\equiv B$, $A\cdot B\equiv A$".

这可用($A=1$, $B=1$), ($A=0$, $B=1$), ($A=0$, $B=0$)三种情况一一加以驗証。

§17 主加法标准形

以邏輯积、邏輯和、否定三种操作能組合成各种各样的邏輯函数;反之,是不是无論哪一个函数都可用这些操作表示呢? 其答案是肯定的。这些函数可用怎样的式子表現,則可直接由眞值表着手来考慮,即在 $f(X_1, X_2, \cdots, X_n)$ 的眞值表中,将 f 为1的各个組合列出,将这些組合的邏輯式表出,并用 $\vee$ 操作联結起来。也就是,在这些組合的 $X_1, X_2, \cdots, X_n$ 之中有些是0,有些是1,将为0的 X_i 写作 X_i',为1的 X_i 写作 X_i,然后从 $i=1$ 开始到 n 写成乘积,就是这一状态的表达式。将眞值表中1的所有各栏取出,并以邏輯和联結,即得这个函数的表达式❶。

将乘积 $(X_1\vee X_1')\cdot(X_2\vee X_2')\cdots(X_n\vee X_n')$ 完全展开(表現为 2^n 项),各项与眞值表中各栏对应,这 2^n 项的总和中,仅仅留下表中等于1的各栏,其他的抛弃,所得的部分和就是 $f(X_1, X_2, \cdots, X_n)$ 的表达式。这个形式称为 f 的**主加法标准形**("积之和"标准形)(principal disjunctive canonical form).

所有的邏輯函数都能写成主加法标准形。

主加法标准形的对偶关系式称为**主乘法标准形**(principal conjunctive canonical form). 这是由 $X_1\cdot X_1'\vee X_2\cdot X_2'\vee\cdots\vee X_n\cdot X_n'$ 的形式利用分配律所形成的 2^n 项的"和之积"式,是保留眞值

❶ 参看表 17.1 及式 (17.1)。——譯者注

表中等于 0 的各项而得。

例如表 17.1 所表达的主加法标准形及主乘法标准形是:

$$X_1'\cdot X_2'\cdot X_3'\vee X_1\cdot X_2\cdot X_3'$$
$$\vee X_1\cdot X_2\cdot X_3 \qquad (17.1)$$

及

$$(X_1'\vee X_2\vee X_3)\cdot(X_1\vee X_2'\vee X_3)$$
$$\cdot(X_1\vee X_2\vee X_3')\cdot(X_1'\vee X_2\vee X_3')$$
$$\cdot(X_1\vee X_2'\vee X_3'). \qquad (17.2)$$

表 17.1

X_1	X_2	X_3	f
0	0	0	1
1	0	0	0
0	1	0	0
1	1	0	1
0	0	1	0
1	0	1	0
0	1	1	0
1	1	1	1

于是有

定理 17.1 有限个邏輯变数 $X_1, X_2, \cdots, X_n$ 的邏輯函数皆可由邏輯积，邏輯和及否定三种操作組合成表达式。

利用[6]和[7a]，[7b]，可得

$$A\cdot B=(A'\vee B')', \qquad (17.3)$$
$$A\vee B=(A'\cdot B')', \qquad (17.4)$$

則所有的邏輯式可由邏輯和与否定，或者邏輯积与否定二种操作組成。但是，仅由邏輯和与邏輯积二种操作是不能組成的，然而各邏輯参数和他們的直接否定 $X_1', X_2', \cdots, X_n'$ 共同用 $\vee$ 和 & 两种操作却可表示各种邏輯式。

主加法标准形不是邏輯函数的唯一表示，更不是簡单的表示。但是，用它作为出发点，再把前述各定理加以簡化，通常可获得更簡单的表示方法。当然，籠統的一般性化簡方法是沒有的，和代数的因子分解相同，对于不同的情况要采取各种不同的方法。不管怎样，依据純粹"形式的变形"，对給定的邏輯函数仍可加以簡化。

例如上面的例子，利用分配律[3a]，得

$$X_1'\cdot X_2'\cdot X_3'\vee X_1\cdot X_2 \qquad (17.5)$$

及

$$(X_1' \cdot X_2' \vee X_1 \cdot X_2) \cdot X_3' \vee X_1 \cdot X_2 \cdot X_3, \qquad (17.6)$$

此时，可看出前式較簡单，但由于其他理由，有时也采取后式[1]。

邏輯函数的主加法标准形，若不計項的順序及各項中因子的順序，則只有一种。不同(不恒等)的邏輯函数由于所得的眞值表不同，他們的主加法标准形也是不同的。于是有

定理 17.2　两个邏輯函数相等的充分必要条件是两个主加法标准形完全一致(不計积的順序及和的順序时)。

此定理是容易理解的。

关于主加法标准形的一个重要定理是，任意邏輯函数的表达式，利用公式[1]～[10]仅使它**形式地变形**，皆可得到主加法标准形。这并不考虑邏輯操作或邏輯函数的具体意义，只是以形式的**公理論**建立邏輯代数，并表示求得眞值表的途徑。本书不以数学基础理論中的邏輯代数为討論对象，故从略。

§18　其他的基本邏輯操作

所有邏輯函数皆可由邏輯和、邏輯积、否定三种，或者由否定与其他二者之一所組成。进一步可考虑如何由一个操作即**唯一操作**来組成所有的函数，这样就簡单了。例如

$$\sigma(A, B) = A' \cdot B', \qquad (18.1)$$

此 σ 操作即是唯一操作。在此，和、积及否定全都被它表示，

$$A' = \sigma(A, A), \qquad (18.2)$$

$$A \vee B = (A' \cdot B')' = \sigma(\sigma(A, B), \sigma(A, B)), \qquad (18.3)$$

$$A \cdot B = (A')' \cdot (B')' = \sigma(\sigma(A, A), \sigma(B, B)). \qquad (18.4)$$

这个 σ 称为 Sheffer 的划函数 (stroke function)。除此以外，也可用函数 $\sigma'(A, B) = A' \vee B'$，其意义是很明显的。

[1] 有时輸入端限制为 2 个时，可采用(17.6)式。——譯者注

此外，下列函数也有上述的性质：

$$\iota(A,\ B)=A\cdot B', \tag{18.5}$$

在此須要应用常数 1.

$$A'=\iota(1,\ A), \tag{18.6}$$

$$A\cdot B=\iota(A,\ \iota(1,\ B)), \tag{18.7}$$

$$A\vee B=\iota(1,\ \iota(\iota(1,\ A),\ B)). \tag{18.8}$$

$\iota(A,\ B)$ 称为**禁止元件**(inhibitor)。

多数决定操作 多数决定操作虽不是唯一操作，但却是一个感兴趣的基本邏輯操作。表 16.1 中的 Y_2 栏是三变数的邏輯函数，从表中可明显看出它的規則，当 X_1, X_2, X_3 的 0 的数目多于 1 的数目时，就等于 0；1 的数目多于 0 的数目时，就等于 1. 这就是称为多数决定函数的緣来。**多数决定函数**写成 $(X_1,\ X_2,\ X_3)$，其式子为

$$(X_1,\ X_2,\ X_3)\equiv X_1\cdot X_2\vee X_1\cdot X_3\vee X_2\cdot X_3 \tag{18.9}$$

$$\equiv(X_1\vee X_2)\cdot(X_1\vee X_3)\cdot(X_2\vee X_3), \tag{18.10}$$

右边二式互为对偶形式，即 $(X_1,\ X_2,\ X_3)$ 这个函数的对偶函数还是 $(X_1,\ X_2,\ X_3)$. 这种函数是**自偶**的。

邏輯和，邏輯积为

$$A\cdot B\equiv(A,\ B,\ 0), \tag{18.11}$$

$$A\vee B\equiv(A,\ B,\ 1) \tag{18.12}$$

是很明显的。因此，使用多数决定操作和否定操作可組成任意的邏輯函数。不仅要用原来的独立变数 X_1, X_2, …, X_n，还要有 X_1', X_2', …, X_n' 和"常数" 0, 1，以作成任意的函数。

在使用多数决定与否定构成的邏輯式中，将其中全部常数 0, 1 互相对調，即将 0 变成 1, 1 变成 0 代入，則結果与以 &, ∨ 分别代替最初函数的 ∨, & 而得的結果相等。因为，原来多数决定式中，含有常数的多数决定函数以常数 0, 1 交换即相当于 ∨, &

变换，不含常数的多数决定函数，∨，& 变换后邏輯式是不变的。因此，"0 与 1 变换代入时，函数即变为相应的对偶函数"。特別是

"不含有常数的，只用多数决定与否定組成的函数是自偶的函数，即满足关系：

$$(f(X_1, X_2, \cdots, X_n))' = f(X_1', X_2', \cdots, X_n')\text{.''} \quad (18.13)$$

其逆

定理 18.1　自偶函数可只由多数决定操作和否定操作組成，不附带常数。

証明　函数 $f(X_1, X_2, \cdots, X_{n-1}, X_n)$ 中最后的 X_n 以常数 1 代替，得 $(n-1)$ 个变数的函数

$$f_1(X_1, X_2, \cdots, X_{n-1}) \equiv f(X_1, X_2, \cdots, X_{n-1}, 1). \quad (18.14)$$

因此 f_1 可由多数决定和否定并借助于常数組成。式中常数 1 出現的地方，全部以 X_n 置换，常数 0 出现的地方全部以 X_n' 置换，则产生 f_0. 当然

$$f_0(X_1, X_2, \cdots, X_{n-1}, 1) \equiv f(X_1, X_2, \cdots, X_{n-1}, 1), \quad (18.15)$$

因为 f 本身是自偶的，所以 f_0 在构造上也是自偶的。将(18.15)两边进行否定后必然为：

$$f_0(X_1', X_2', \cdots, X_{n-1}', 0) \equiv f(X_1', X_2', \cdots, X_{n-1}', 0). \quad (18.16)$$

然而，$X_1', X_2', \cdots, X_{n-1}'$ 等只不过是变数的記号，两边同时写成 $X_1, X_2, \cdots, X_{n-1}$ 沒有什么不同，因此，(18.15)与(18.16)是相等的，f 和 f_0 实质上是同一个函数。f_0 只以多数决定与否定操作組成，不包含常数。于是定理得証。

多数决定是变参元件的基本操作。关于各种邏輯函数以多数决定操作实行的方法，后面有很多实例。

§19 单調函数

一个邏輯函数 $f(X_1, X_2, \cdots, X_n)$，当 $X_1 \subset Y_1$, $X_2 \subset Y_2, \cdots$, $X_n \subset Y_n$ 时，常有如下的性质：

$$f(X_1, X_2, \cdots, X_n) \subset f(Y_1, Y_2, \cdots, Y_n),$$

則称 f 为**单調函数**(monotonic function). $A \cdot B$, $A \vee B$ 就是A, B 的单調函数，且有

"由任意变数仅以 $\vee$, & 两种操作組成(不含有否定)的函数，总是单調的。"

反之，下面的定理成立：

定理 19.1 $X_1, X_2, \cdots, X_n$ 的单調函数 $f(X_1, X_2, \cdots, X_n)$，总可以只由 $\vee$ 及 & 联結其变数組成。

这定理容易由数学归納法証明。由单調的定义知

$$f(X_1, X_2, \cdots, X_n, 0) \subset f(X_1, X_2, \cdots, X_{n-1}, 1). \quad (19.1)$$

因为

$$f(X_1, X_2, \cdots, X_{n-1}, X_n) \equiv f(X_1, X_2, \cdots, X_{n-1}, 0) \vee f(X_1, X_2, \cdots, X_{n-1}, 1) \cdot X_n, \quad (19.2)$$

即 $(n-1)$ 个变数的邏輯函数

$$f_0(X_1, X_2, \cdots, X_{n-1}) \equiv f(X_1, X_2, \cdots, X_{n-1}, 0)$$

及

$$f_1(X_1, X_2, \cdots, X_{n-1}) \equiv f(X_1, X_2, \cdots, X_{n-1}, 1)$$

可用邏輯和、邏輯积操作組成 n 个变数的函数。f_0, f_1 仍是单調函数极为明显。同样逐漸減少变数，則全部可由 &, $\vee$ 操作組成。

上述以 $\vee$, & 的操作的組合表达单調函数的方法，若变数引出的順序一定，則它也只有一个确定的形式，这样得到的邏輯式可看作是标准形式。

同样，可利用下式分解，所得的是与上式有对偶关系的标准形

式，

$$f(X_1, X_2, \cdots, X_{n-1}, X_n)=(f(X_1, X_2, \cdots, X_{n-1}, 0)\vee X_n)$$
$$\cdot f(X_1, X_2, \cdots, X_{n-1}, 1). \qquad (19.3)$$

同样，对于多数决定的操作，下面的定理成立。

定理 19.2　$X_1, X_2, \cdots, X_n$ 的邏輯函数 $f(X_1, X_2, \cdots, X_n)$，仅由其变数和常数 0，1 用多数决定操作組成(不用否定操作)的充分与必要条件是，这个函数对其变数是单調的。

因为多数决定操作的函数是单調函数，这是很明显的，而这也就是必要条件。充分条件的証明，与上面相同，若 f 为单調，则

$$f(X_1, X_2, \cdots, X_{n-1}, X_n)$$
$$=(f(X_1, X_2, \cdots, X_{n-1}, 0), f(X_1, X_2, \cdots, X_{n-1}), X_n). \qquad (19.4)$$

上式可用多数决定操作归納为 $(n-1)$ 变数的函数，以此类推卽可証明。

§20　邏輯代数的代数化

在邏輯代数的規則中，邏輯积服从普通代数的法則，否定可写成下列代数式：

$$A'=1-A. \qquad (20.1)$$

应用此关系，則所有的邏輯式可写成代数式

$$A\vee B=(A'\cdot B')'=1-(1-A)(1-B)=A+B-AB, \qquad (20.2)$$

$$A\cdot B\vee A\cdot C\vee B\cdot C=AB+AC+BC-2ABC \qquad (20.3)$$

等，这里使用了 $A^2=A$ 的性质。

檢査两个邏輯式是否相等，方法之一是利用多項式比較。因此

定理 20.1　邏輯函数用多項式表示时，若各项中作为因子的各变数只出現一次，則这个邏輯函数的表示式是唯一的。因此，两

个邏輯函数的多項式表示的相等，是两个函数相等的必要与充分条件。

証明 多項式表示式将邏輯函数唯一确定，这是明显的(因可确定眞值表)。反之，同一个邏輯函数，若有两种多項式 P, $\bar{P}$ 表示，則 $P-\bar{P}$ 当变数 X_1, X_2, $\cdots$, X_n 以 0 及 1 的各种組合代入时必須是 0. 将 $P-\bar{P}$ 写成下式：

$$P-\bar{P}=QX_n+R, \tag{20.4}$$

其中 Q,R 是 $X_1,X_2,\cdots,X_{n-1}$ 的多項式，这里为了使以 $X_n=0$ 或 $X_n=1$ 代入后都为 0 起見，Q, R 以 $(n-1)$ 个变数的所有 0 和 1 的組合代入必須为 0. 假如定理在 $(n-1)$ 个变数的情况下成立，Q, R 都必須恒等于 0，因此 n 个变数时定理一定成立，而一个变数时定理成立是明显的，利用数学归納法，定理得証。

多項式表示式不仅将邏輯式表示成代数式，当輸入变数是**独立随机变数**时，以 X_1, X_2, $\cdots$, X_n 分别表示这些变数出現 1 的概率，則邏輯函数的多項式表示式的数値，就是那个函数出現 1 的概率。

代数化的另一个方法是采用新操作 $\oplus$. $\oplus$ 是 mod 2 加法，有

$$A\oplus B\equiv A\cdot B'\vee A'\cdot B=A+B-2AB \tag{20.5}$$

的意义。反之，

$$A\vee B\equiv A\oplus B\oplus AB, \tag{20.6}$$

$$A'=1\ominus A. \tag{20.7}$$

因此，所有操作，象代数的乘法可由 mod 2 的加法組成。实际上，将前述的多項式表示式用 mod 2 簡化，除去偶系数的各項，奇系数各項的系数改为 1，則所得的多項式即是用这种方法表示的多項式。

实际上物理的邏輯操作元件，在很多場合可以实現某种代数关系式，所以，这种表示也还有用。

§21 邏輯操作的物理实現 I

叙述有关邏輯函数的目的，就是便于以后説明实际执行这些操作的計算机。因为数字的計算就是一种邏輯操作，計算的結果就是那些数据的邏輯函数。例如加法

$$1+2=3,$$

可以由下面的邏輯式表示：

$$[x \text{ 是 } 1]\cdot[y \text{ 是 } 2]\subset[x+y \text{ 是 } 3],$$

反之，数字計算机能进行邏輯函数的計算(参照§20以代数式写出邏輯函数)。

机器实現邏輯操作，須使用物理現象中对应的邏輯关系❶。数字計算机有机械的(狭义地説)，有使用电触点性质的电气机械的，也有使用电子的等。实际上，説明他們在电子計算机中的应用，通过具体的对象(image)有助于理解抽象的邏輯代数。

邏輯操作都是用簡单的基本元件按照特有的組合方法組合而成，有的場合是用元件本身，也有另外的場合是用組合为基础，各与基本的邏輯操作相对应。

机械的邏輯元件 設計执行邏輯和，邏輯积的机械元件是极容易的。图 21.1 中两块板上面放一块板，下面的板可上下移动，上面的位置对应 1，下面对应 0. 上面的板被托上的位置 (0, 1) 是下面板的位置(邏輯变数)的邏輯和。相反，当規定下面位置为 1，上面为 0 时，得邏輯积。

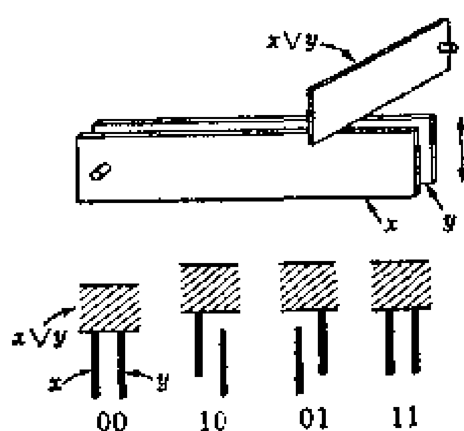


图 21.1 机械的邏輯机构

代替板的上下运动，可用沿板

❶ 在这种意义上，数字計算机也是一种模拟的机器。

长的方向作水平运动，只要在板上設計出适当的缺口。以缺口移过来的位置作为1，即可得邏輯积操作。使用几块有缺口的横板，可得多变数的邏輯积(图21.2)．这类机构常用于电傳打字机及各种鎖钥。这种被托上的构造是邏輯操作的典型的机构。

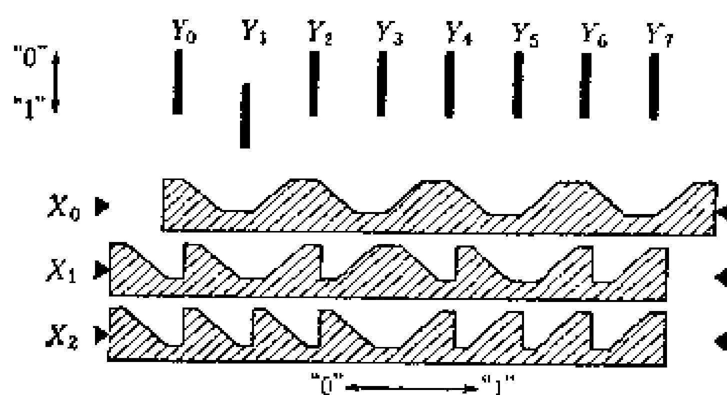


图21.2　利用横向移动的邏輯机构

将这种构造分2級使用，就能得到"邏輯积的邏輯和"，因而加法标准形以至任意的邏輯函数皆能实现。

机械的邏輯元件有各种各样的构造，其特点非常简单，在各种机械和工具上都曾使用过。

电触点与继电器　将开关电路按不同方式組合，可以构成計算机。普通的开关，有用手撥动的，也有以电磁鉄自动撥动的．那种以电磁鉄撥动的开关称为**继电器**，将綫路切断或接通的部分称为**触点**(contact)．

利用触点的計算机的特点是，将触点**串联**，**并联**組合成各种复杂的邏輯函数，非常方便，每个继电器都当作邏輯变数．例如，X_1这个继电器的**前触点**(make contact)，是由磁鉄吸引时接通的触点，在这样接通的状态下，邏輯变数$X_1=1$，而$X_1=0$是指触点离

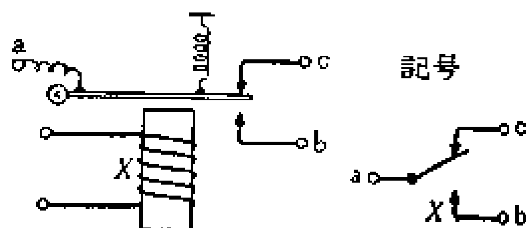


图21.3　ab 間：前触点
ac 間：后触点

开的状态。**后触点**(break contact)是磁铁切断状态下接通的触点，作为X_1的否定X_1'。同一个継电器可以有很多触点同时动作，这些触点总是认为代表同一个邏輯变数(包括否定)的。

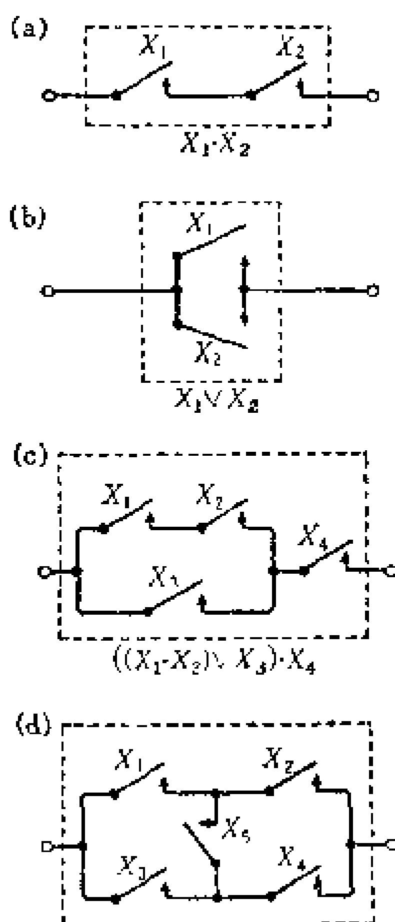


图　21.4

以这些触点的串联、并联組成复杂的綫路，并将这綫路的两头取出，而只考虑这两点間导电与否，并认为两点間导电时函数值为1，不导电时函数值为0，这函数称为这两点間綫路(两端网絡)的**傳导函数**，它代表以各触点为邏輯变数的函数。

作为簡单的情况有

(i) 以X_1，X_2代表两个触点的串联，其傳导函数为Y，

$$Y \equiv X_1 \cdot X_2.$$

(ii) 以X_1，X_2代表两个触点的并联，其傳导函数为Y，

$$Y \equiv X_1 \vee X_2.$$

将这种触点的組合当作一个触点，将它們再組合，可产生更复杂的两端网絡。例如图21.4(c)就是$(X_1 \cdot X_2 \vee X_3) \cdot X_4$这个邏輯函数。以邏輯积，邏輯和組合构成的邏輯函数所对应的綫路，总是串联，并联的組合构成。若使用后触点X_1'，X_2'，…来构造，应用主加法标准形(或主乘法标准形)可作成对应于任何复杂的邏輯函数的触点网絡，图21.5就是对应于表16.1的Y_2栏的綫路。

实际上，主加法(或主乘法)标准形，在很多情况下决不是最簡单的实现方法。象邏輯式依据各种法則来簡化一样，触点綫路也可使用相应的法則轉換成比較簡单的等价的綫路，例如图21.5

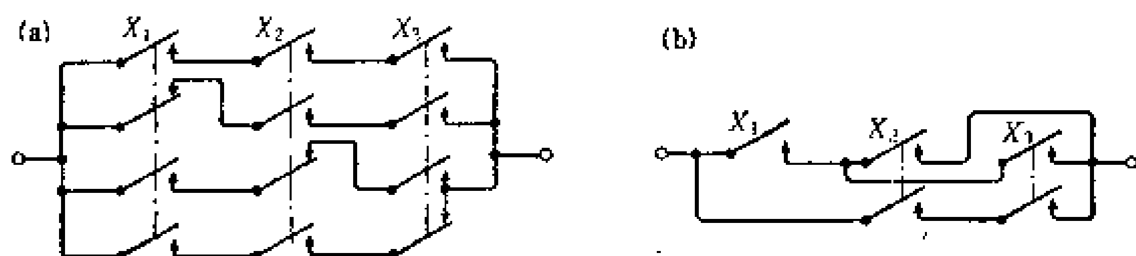


图 21.5 同一个继电器有四个触点上下联动

的(a)綫路与(b)綫路是等价的。

除此以外，触点綫路还有不能看成是由串联、并联結合的复杂的組合，图 21.4 的(d)就是一例，称为桥路。它的邏輯式如下：

$$Y \equiv X_1 \cdot (X_5 \cdot X_4 \vee X_2) \vee X_3 \cdot (X_5 \cdot X_2 \vee X_4),$$

但是，表达这个式子的綫路构造却是比較簡单的。

給出一种形状的邏輯函数，将其傳导函数的触点网絡造出，称为**綜合**(synthesis)。虽然它在实用上是很重要的，但是系統地綜合优良綫路的方法至今尚未找到。

給出邏輯式，以 $\vee$ 与 & 交換，获得其对偶邏輯式，若触点网絡能在平面上画成不重迭交叉的綫路❶(称为"能展开的")，一般有**对偶变换**存在。因此，

对偶变換規則 若一个邏輯函数 $f(X_1, X_2, \cdots, X_n)$ 用在某一平面能展开的触点网絡 N 表示，将其对偶函数

$$f^*(X_1, X_2, \cdots, X_n) = (f(X_1', X_2', \cdots, X_n'))'$$

表达的网絡記为 N^*，則 N^* 可如下获得：对于 N 网絡中所有的触点，画出横跨它的触点即为 N^* 的触点；若 N 触点网絡形成一个封閉的 m 角形，則在 N^* 中，形成 m 个节点(node)；对应于 N 的两个輸入端的联綫有横穿过它的 N^* 的两个輸入端的联綫(图 21.6)。

証明从略，但为了帮助思考，下面作一些說明。設想 N 网絡在平面上建筑了墻壁和門，N^* 表示人通过这些門的通路，門关着，人

❶ 即沒有两条綫联結，按一个触点考慮。

就不能通过。这与上述定理是相同的。这样得到的 N 与 N^* 的对偶性质是拓扑学对偶性的一种。

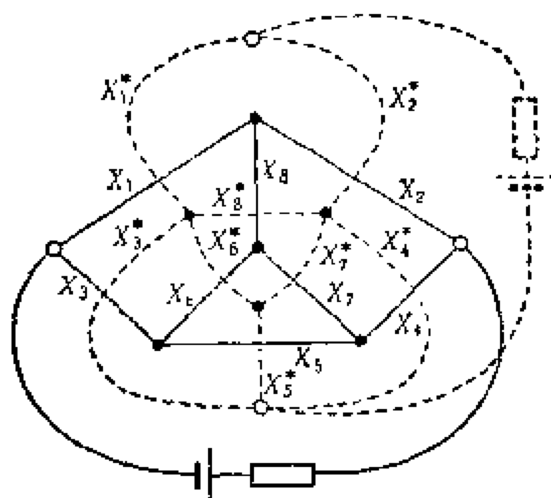


图 21.6　对偶网絡
实綫：原网絡
虚綫：它的对偶网絡

继电器　一个邏輯函数以相应的触点綫路組成时，在較复杂的場合，可使用继电器。如有完全相同的部分电路，或有好几处表示它的否定的电路，可将这些电路拖动继电器的磁鉄，而以那个继电器的前触点和后触点代替这些电路及其否定电路来使用。长的邏輯式，可将式子的一部分代以新的变数，写上某个符号，再写下此符号的定义式，从而将这操作分成几重进行。

使用继电器，将产生时間延迟。結果，由触点綫路构成的計算机速度較慢，这是由继电器的操作时間所引起的。

日常也使用触点綫路，如在两个地方装着电灯开关，两处的触点是并联的，要想关灯，非两方切断不可，这是邏輯和的場合。反之，图 21.7 为三个开关 A, B, C 随处都可关灯。这里一再地应用了 §16 的表 16.1 中 Y_1 的函数。

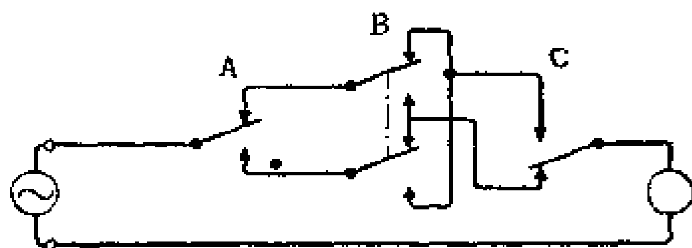


图 21.7 三个地方随处关灯的开关

§22 邏輯操作的物理实現 II（电子元件）

1) **二极管**　在电子計算机中的邏輯变数及函数，常以端点的

电压来表现,例如,端点对地 +10V 作为 1 状态,0V 作为 0 状态,这种状况与机械的情况相似。

在机械的邏輯元件中,若板 B 托在板 A 上,則 B 板可以朝比 A 高的方向上升,而不能朝比 A 低的方向下降。这个性质,在二极管整流器中是相同的。

在二极管中,电流由 A 端(阳极)流向 B 端(阴极),逆向不流。当 B 的电位比 A 高时,电流不从 B 流向 A,但是,当 B 点的电位比 A 点低时,电流就流通,并阻止 B 点电位下降。若是 B 的电位降低的力弱,即在它的内阻抗高的情况下,B 的电位不可能比 A 低很多。因此,在二极管中,設想正电位相当于机械装置中的上面位置,则阴极电位比阳极的高或两者相等的情况,相当于(以机械比拟)阴极托在阳极上的情况,正确地說,电流流通时,阴极的电位比阳极稍低一点。因此,上面所述是理想的情况。

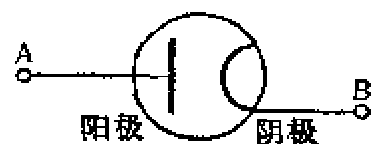


图 22.1 二极管符号

以两个(或更多的)二极管的阴极連通作为輸出端,将各阳极作为輸入端,就成为邏輯和的綫路。只要任一輸入端加上正电位,輸出端即为輸入端之中的电位最高的正电位。

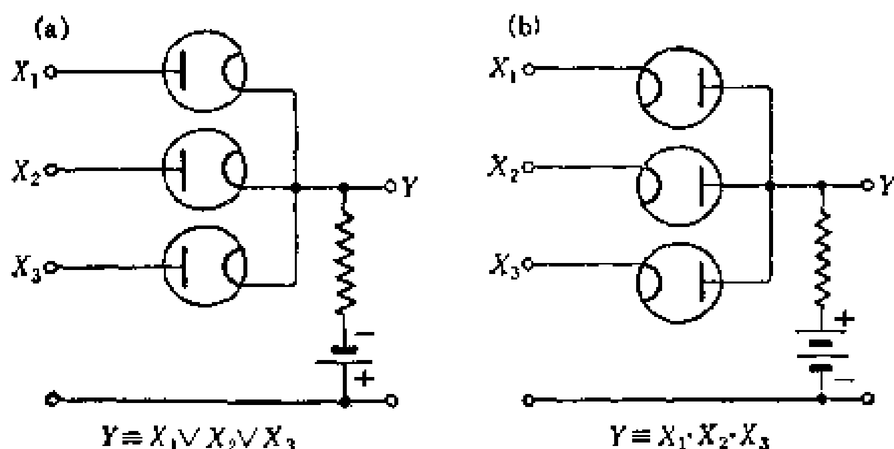


图 22.2 二极管的邏輯和及邏輯积

将两个以上的二极管的阳极連通作为輸出端，各阴极作为輸入端，这是邏輯积的綫路。若輸入端之中有一个零电位，即使其他是正电位，由于輸出端是輸入的最低电位，所以也是零电位。

这种邏輯元件，理論上能以多級重复，构成复杂的邏輯函数。实际上，多級重复时，电压要减小。使用2級以上的时候，中間就要加入放大器，使电压达到規定的数値。

除此以外，还可使用三极管、多极管等，但此处从略。

2) **变参元件** 以强磁性材料为磁芯作成綫圈，再与电容組成諧振回路，这是变参元件的主体。它是与前述完全不同的邏輯元件。它的原理从略。在变参元件中，0及1是由相位相反的正弦交流电代表的。此諧振回路中电流是

$$\text{“0”状态} \qquad -I_0\cos\omega t,$$

$$\text{“1”状态} \qquad I_0\cos\omega t.$$

将 n 个变参元件跟一个下一級的变参元件弱結合，则其輸入电流为：

$$I_{\text{入}}=\sum_{i=1}^{n}\varepsilon_i k I_0\cos\omega t, \quad |k|\ll 1,$$

此处 ε_i 为第 i 号变参元件的状态，为0时则 $\varepsilon_i=-1$，为1时则 $\varepsilon_i=1$.

变参元件有**放大作用**。将上述弱小的輸入信号放大，輸出为

$$I_{\text{出}}=\pm I_0\cos\omega t,$$

它与輸入振幅无关。这里是‘+’还是‘−’由 $I_{\text{入}}$ 的符号决定。故

$$I_{\text{出}}=\operatorname{sgn}\left(\sum_{i=1}^{n}\varepsilon_i\right)\cdot I_0\cos\omega t.$$

若 n 为**奇数**，则 $I_{\text{出}}$ 的相位由前級变参元件輸出相位按照**多数决定**的原則确定。n 为偶数时，輸入为0，这时輸出不确定。原則上 n 只取奇数，实际 $n=3$（有时用 $n=5$）. 这种变参元件的邏輯操作是以§18的多数决定操作为基础的。

变参元件的特性之一是，从一个变参元件向下一級变参元件

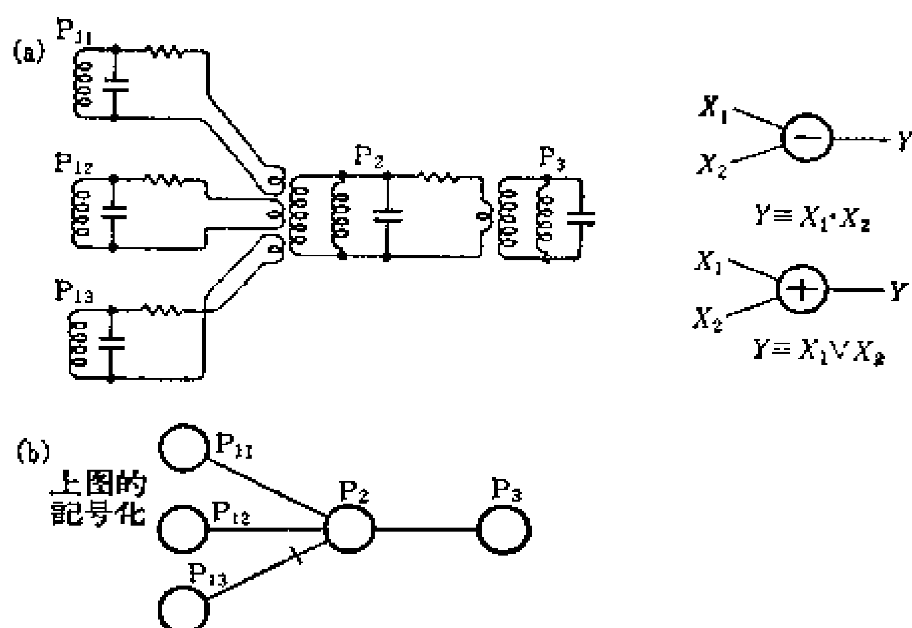


图 22.3 变参元件的結合法

送一次信号,要有一定的时間延迟。因此,在变参元件的邏輯綫路中,若要使信号先分开,然后合起来,則双方經过的級数非相等不可。

在变参元件中,否定的操作是簡单的,仅在結合的地方將变压器綫圈极性反一下即可。

使用变参元件时,由多数决定及否定操作組合,可构成任意的邏輯函数。变参元件組合(变参元件綫路)的图示法,是以圆圈表示变参元件,以綫条表示結合,信号自左向右傳送。含有否定时,在結合綫上画一条短的横綫。另外,"常数" 0, 1 画在图上太繁杂,只在圆中写上+或－来表示。故而,若从左边有两根綫輸入而写⊖,則是表示邏輯积,写⊕則表示邏輯和。

作为邏輯函数构成元件的变参元件是特别单純的。因为,变参元件本身,是三个輸入端和一个輸出端的非常簡单的計算机元件,用它可組合成复杂而高級的計算机,一个变参元件的輸出端是其他变参元件的輸入端,沒有比这更簡单的方法了。

下面是变参元件网絡的构成規則。变参元件的一个輸入端不

允許和两个以上的其他变参元件的輸出端相联。另外，同类的輸入端，同类的輸出端相联是沒有意义的。但是，一个变参元件的輸出端，可和若干其他元件的輸入端相联(实际上，**分支数**由于技术以及物理的原因而受到限制。但在本編中假設它不受限制)。因此，因果关系是单向进行的，即从輸出端出来朝下一元件的輸入端方向进行。

为了說明方便起見，而且实际在日本广泛使用变参元件的計算机，所以以下几节說明具体的綫路时，主要是說明采用变参元件的实現方式。特别，所有的变参元件都是实际輸入端为两个⊕或⊖的情況，可只用邏輯和与邏輯积元件构成邏輯网絡。

§23 三变数函数

在前述的一般基本邏輯操作 ∨，&，多数决定等的組合来实現邏輯函数方面，將再举几个实际上广泛使用的典型代表，它們就是电子計算机的运算器及控制器的主要部分。

邏輯函数的对称变换 一般 n 个变数的邏輯函数，其变数值 0，1 的組合有 2^n 种，而函数值可选择 0 或 1，因此，眞值表的可能数目有 2^{2^n} 种。即 n 个变数的邏輯函数共有 2^{2^n} 个不同的函数。由表 23.1 可知，当 n 增加时，数字激剧增加；当 $n=4$ 时已不易将其全部表述。因此就需要考虑将其中相似的函数汇总起来，而以少数函数来代表全体函数的問題。这就要考虑对称性。

表 23.1

n	2^{2^n}	眞正的 n 个变数的函数的数目	不同族的 n 变数函数数目	不同族的眞正的 n 个变数的函数数目
0	2	2	1	1
1	4	2	2	1
2	16	10	4	2
3	256	218	14	10
4	65536	64594	238	224

（i）**变数置换** 給定邏輯函数，将其中变数进行置换，一般产生新的函数，一个函数經过某种置换后可以和另一函数相等。因此一类置换对应着一族經置換后可相等的函数，将这些函数作为同族的函数，无对称性的函数，共 $n!$ 个函数构成一組。含对称性的函数的数目較前者少，都包含在一組内。

（ii）**变数的否定** 在有的邏輯函数中，将变数 $X_r, X_{r'}, \cdots$ 否定，变为 $X_r', X_{r'}', \cdots$，也得到新的函数，亦将这类函数当作同族的，屬于否定的同族。

（iii）**函数的否定** 将邏輯函数本身否定，可与原函数作为同族。

将同族函数作为一类，則表 23.1 上函数的数目就变得很小了，不同族的函数数目列在表 23.1 右栏中。

更进一步，在表中記录的 n 变数的栏中，实际上，不是其中的全部变数都表現出来的，事实上也含有变数数目比 n 少的函数，因此真正的 n 变数的函数的数目要在 n 变数栏的数中扣去 $(n-1)$ 变数栏的数目❶。

以下叙述变数数目不大于三的情况，以及他們的实現方法。

二变数函数 一变数沒有問題，所以叙述二变数。它可用下列两个作为代表：

Π_1（門） $A \cdot B$，

Π_2（二进計数器） $A' \cdot B \vee A \cdot B' \equiv A \oplus B$

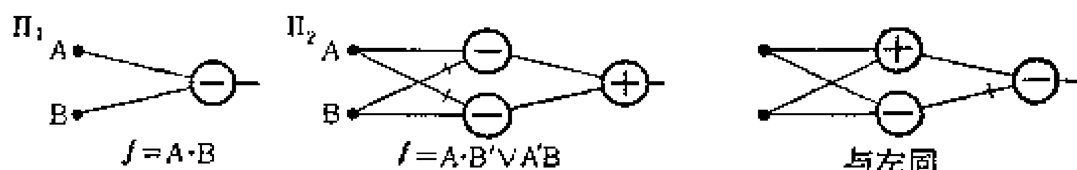


图 23.1

❶ 这段意思是，同族概念引入后，函数数目减少了，类似于变数减少了一个，例如由于引入置换关系后，某些变数失去了独立性。——譯者注

$A\vee B, A\vee B'$,等等是 II_1 的同族函数。在变参元件中,II_1 是1級,II_2 是2級組成的(图23.1). 在图中将⊕讀作邏輯和元件,⊖是邏輯积元件,—+是否定元件,它們都作为普通邏輯元件使用。

II_1 中的邏輯积綫路普通称为**門**(gate),执行开閉信息通路的任务。A 是信息的信号,B 是控制信号,使 $B=1$,則輸出等于 A;$B=0$,則輸出等于0,即切断信息。

属于 II_1 的邏輯和 $A\vee B$ 也同样执行門的任务,用于二个信号的混合。

图23.2表示属于 II_1 的8种同族函数。

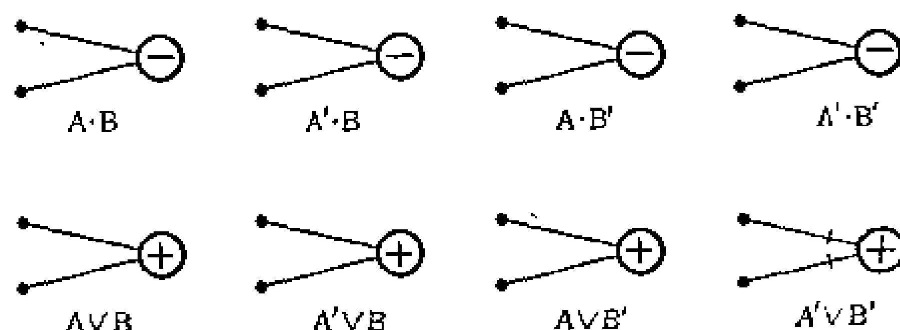


图23.2 $A\cdot B$ 的同族邏輯函数

II_2 中,$A\oplus B$ 称为**二进計数器**(binary counter),是二进計数电路及二进加减电路的基础,也是檢查二个信息的同一性的**比較器**(comparator)的基础。

三变数函数 这里有10种不同族函数。用表和式子表示,以立方体的8个角对应变数的8种組合,函数值为1的地方画一个小球,图示的对称性是一看就明白的。图23.3将这种图与变参元件对照。

其中有些綫路的用途不大,特别重要的有:

III_1 $A\cdot B\cdot C$(**三重一致綫路**) 以二个控制信号控制信息傳送的門或三个信号結合等各种用途。

III_7 $(A, B, C)\equiv A\cdot B\vee A\cdot C\vee B\cdot C$(**进位綫路**) 此处以变

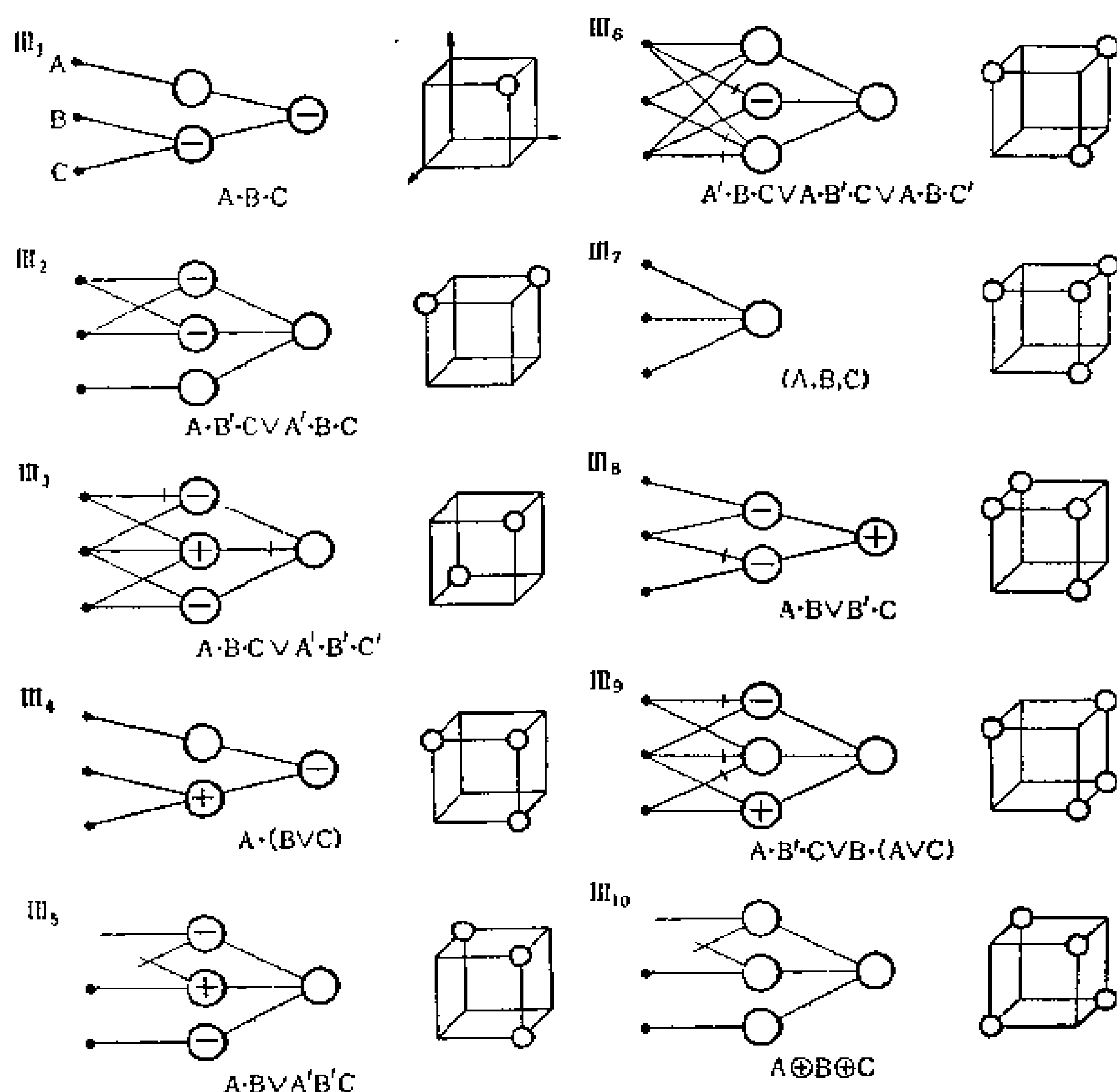


图 23.3 三变数邏辑函数的变参元件綫路

参元件构成很簡单，若以普通 ∨，& 构成将很麻煩。主要用于二进加法器的进位。

III_8 $A\cdot C\vee B\cdot C'$ （**双擲开关**） 以 C 作为控制信号，交替地让 A, B 二信号通过。

III_{10} $A\cdot B\cdot C\vee A'\cdot B'\cdot C\vee A'\cdot B\cdot C'\vee A\cdot B'\cdot C'$ （**二进加法器**） $A\oplus B\oplus C$ 綫路，沒有常数輸入，是自偶函数。象 §18 定理 18.1 的証明一样，在二变数的非自偶函数 II_2 中，将常数变成变数即可

得到。

III_3 $A\cdot B\cdot C\vee A'\cdot B'\cdot C'$（**全有或全无**）用于檢查三个輸入全部为1或0的綫路。这个函数若以普通的标准形构造，則要二个III_1綫路，再取其邏輯和，至少有7个元件，此图中电路要多三个元件，并且3級构造使延迟时間增长。图中的綫路不易一看就理解它的操作，可用邏輯式驗算。

III_8 $A'\cdot B\cdot C\vee A\cdot B'\cdot C\vee A\cdot B\cdot C'$（**三中取二的綫路**）是檢查三个輸入中仅有二个1及一个0的綫路，用于誤差檢查等綫路。

§24 多变数函数

四变数以上的函数，为数甚多，用多数决定元件組成的場合，往往成为多于3級以上的复杂构造。这里不作一般的叙述，仅就实用上重要的若干例子記述如下：

（i）**选擇輸入綫路** 有N个信息輸入端及n个控制(选擇)輸入端的綫路，按照不同的控制輸入組合，从N个信息輸入中选擇一个信号，在輸出端表現出来。此即前节III_8的一般化。

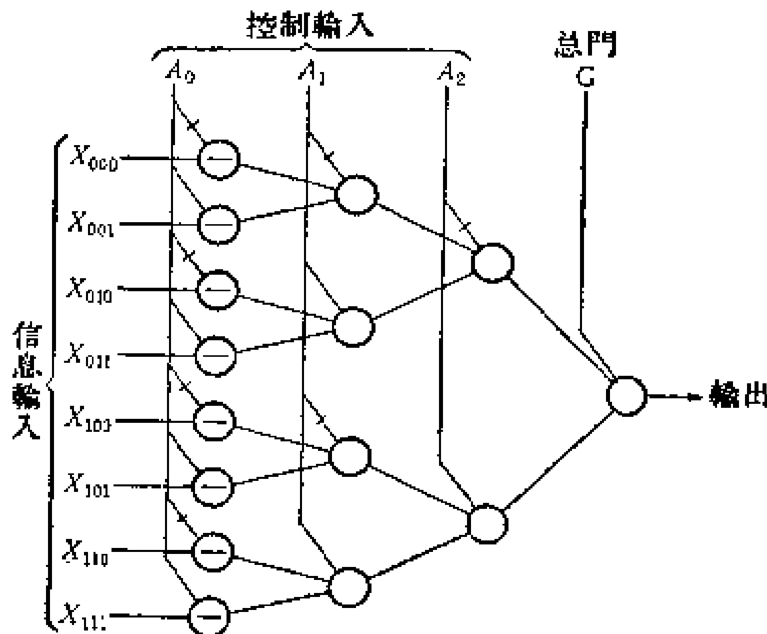


图24.1 选擇輸入綫路

考虑典型的 $N=2^n$ 的場合，卽根据 n 根控制輸入綫的全部 2^n 个組合，选擇不同的信息輸入。它在計算机的存貯器等方面有各种不同的应用。图 24.1 的綫路是树状(tree)綫路，第 2 級以下使用不加常数輸入的三輸入多数决定元件。三个輸入中，从左边輸入的二个必有一个是“0”信号。因此这多数决定元件不加常数輸入卽可作为門使用。

信息輸入端数不是 2 的方冪时，有一部分分枝可以省略。

控制信号可不用二进碼而用“N 中取 1”碼表示，卽控制信号綫数与輸入信号綫数相同，在控制信号綫中只有一个信号时为“1”，与之相应的輸入信号綫卽被选中。选擇信号由譯碼器(参看§25)送来。这时，綫路只是 n 个輸入与各个控制信号以 & 結合，其輸出再以 ∨ 混合，在混合处要有前述选擇綫路同样多的元件，那是不利的。

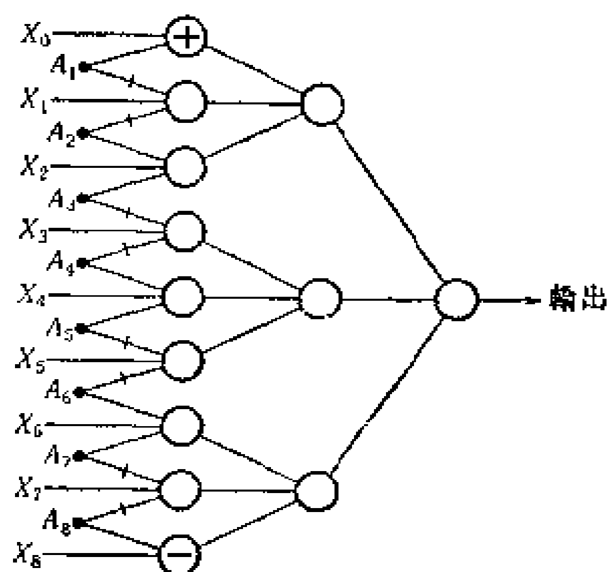


图 24.2 三分支选擇輸入綫路

用“阶段型”选擇信号，卽选擇信号的 N 个輸入端中，从第 1 到第 r 端輸入“1”，第 $(r+1)$ 号以后都輸入“0”、以这样的方式选擇輸入信号綫中第 r 号而輸出，电路示于图 24.2. 信号混合时一次只有三个輸入，元件及延迟都减少了。三个輸入中，二个互相抵消，

殘留一个让信号通过。这种选擇信号的輸入最好使用不等式譯碼器(§25, ii)的輸出。

(ii) **加法标准形的一般邏輯綫路** 在前述选擇輸入綫路中，假如在信号輸入端加入各个适当的常数，而选擇信号对常数选擇，可产生任意的 n 变数的邏輯函数。这个綫路是以$(n+1)$級构成的，从左边加入的常数不必全部都加在第一級上，而其結果是 n 变数的邏輯函数要由 n 級构成。輸入中只有二个变数在第一級輸人，其余$(n-2)$个变数，分别向 2 級，3 級，…，$(n-1)$級輸入，輸出前的延迟可以减少。图 24.3 就是用这个方法构成的四变数加法綫路，标准形构成后，能共用的地方就共用，这样可以簡单化。

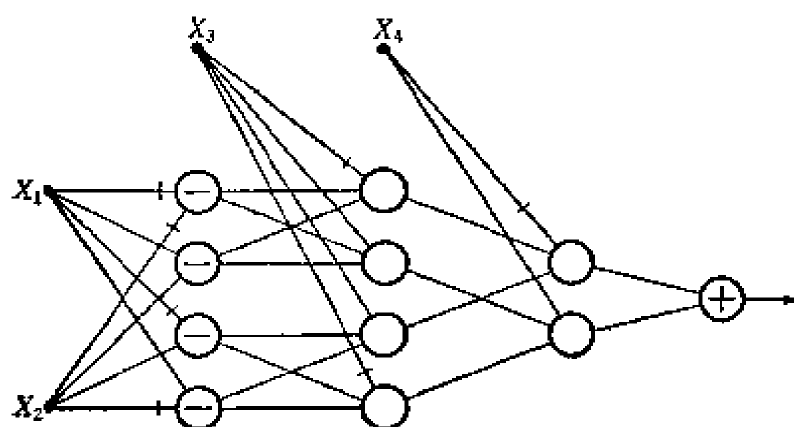


图 24.3 以加法标准形构成的 $X_1 \oplus X_2 \oplus X_3 \oplus X_4$ 的綫路

这种构造，是以加法标准形为基础的一般形式，但在不少情况下需要很多的元件和級数，例如三变数在这种方法下，大部分的函数要 3 級构造，实际上，应用前面的类型可以全部用 2 級构成。图 24.3 的綫路，可用图 23.1 的 II_2 的 2 級綫路构成，这时元件数目可减少。但本綫路的特征是从 X_3, X_4 到輸出端之間的延迟很小。

(iii) **比較綫路** 有两組 n 个輸入端，这些輸入$(X_{n-1}, X_{n-2}, \cdots, X_1, X_0)$；$(Y_{n-1}, Y_{n-2}, \cdots, Y_1, Y_0)$看成是两个二进数 X, Y. 綫路当 $X>Y$ 时，輸出 1, $X \leqslant Y$ 时，輸出 0. 这个綫路用多数决

定元件是极容易获得的(图24.4)．即 $X_{n-1}>Y_{n-1}$ 輸出必定是1，$X_{n-1}<Y_{n-1}$ 时必为0，$X_{n-1}=Y_{n-1}$ 时，輸出元件 P_{n-1} 上下两个輸入抵消，由前級元件 P_{n-2} 决定，即檢查 X_{n-2}，Y_{n-2} 以下各位，P_{n-2} 与 P_{n-1} 是以同样方式决定的。所以，假如上一位相等时，顺次檢查下一位，可以确定这两个二进数的大小，最低位 X_0，Y_0 比較时，以常数0(－)置入，当 $X_0=Y_0$，从而 $X=Y$ 时，作法与 $X<Y$ 相同。

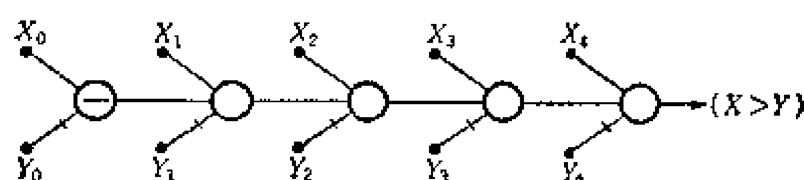


图24.4 大小比較綫路

若 P_0 上置入的常数是"1"(＋)时，就成为判断 $X\geqslant Y$，$X<Y$ 的綫路。

当理解了判断 $X>Y$，$X\geqslant Y$ 的方法后，将两組綫路組合，则輸出是檢查**等号** $X=Y$ 的綫路。若以檢查各位相等的綫路(§23，II_2)組合构成时，本綫路与之相比，可以少用 $(2n+1)$ 个元件。

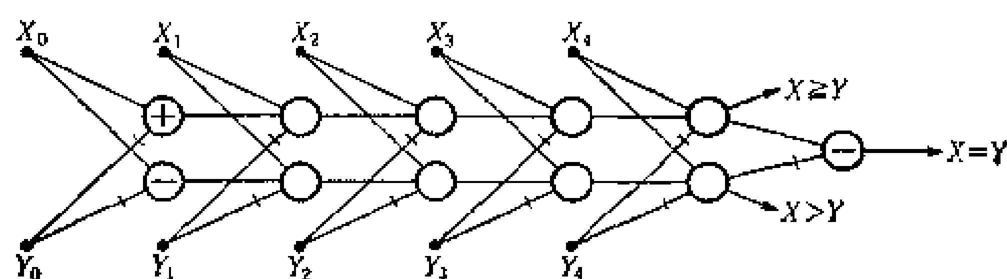


图24.5 等号比較綫路

图24.5的綫路中，輸入要在前后各級上分别加入，因此将計数綫路(参看§30)的輸出送入是合适的；而对并联的輸入则不合适。这时从 X_0，Y_0 至第 $n+1$ 級的延迟是很大的。图24.6是以同样个数的元件組成而延迟最小的比較綫路，一般当 $n\leqslant 2^{\nu}-1$

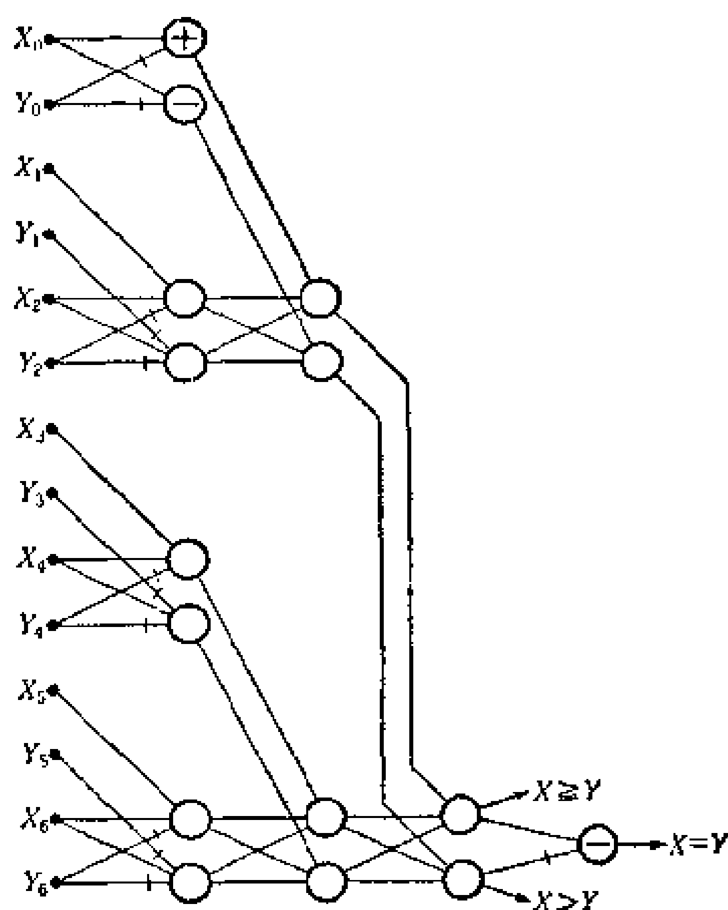


图 24.6　延迟相同的比較綫路

时，只要 $\nu+1$ 級就可得到 $X=Y$ 的輸出。在第 ν 級得到 $X \geqslant Y$ 及 $X>Y$.

此外，当 $X=Y$ 时，已經不限于二进位数的意义而是 $X_i=Y_i$ $(i=0,\ 1,\ \cdots,\ n-1)$，即各位都是相等的。綫路的构造不取各位相等，但結果与各位相等并不矛盾。

§25　多輸出綫路

从若干輸入組成两个以上的函数，可以只討論各个单独的函数的組成綫路。但是，对于不同函数的綫路往往其中有一部分可以公共使用。在单独研討一个个的邏輯函数的綫路之外，仍有必要提出多輸出綫路的設計問題。

（i）**譯碼器**　譯碼器是根据 n 个輸入信号的組合情况，在 2^n

个(或者少一些)輸出端中选擇一个，使其輸出信号为1("2^n中取1"的綫路)。各个輸出是輸入X_1, X_2, …, X_n之中若干为原状其余为否定的邏輯积。故$n\leqslant 2^\nu$时，可由ν級邏輯积树状綫路組成。因为2^n个輸出的綫路有公用的部分，故可簡单化。在图25.1中，先由两个輸入获得"4中取1"的輸出，然后将两組組合，获得"16中取1"的輸出，以下相同。这个綫路是曾經提到的选擇輸入綫路、存貯器写入地址选擇綫路以及下文的函数表等的必要的基本綫路。

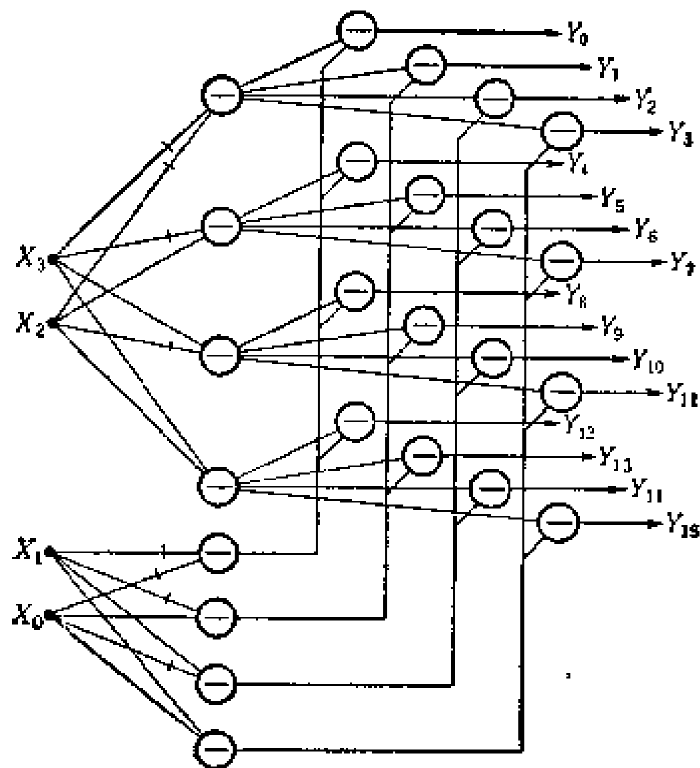


图25.1 譯 碼 器

(ii) **"阶段型"譯碼器** n个輸入端对应于2^n-1个輸出端，輸入X_1, X_2, …, X_n表示二进数X，第r号輸出是当$X\geqslant r$时为1，$X<r$时为0，称为基于不等式的譯碼器。这种譯碼器用比較綫路的集合来实現(图25.2)。

这种譯碼器是图24.2的綫路的必要部分，其他还有很多用途。

(iii) **編碼器** 此綫路有N个輸入及m个輸出，N个輸入中

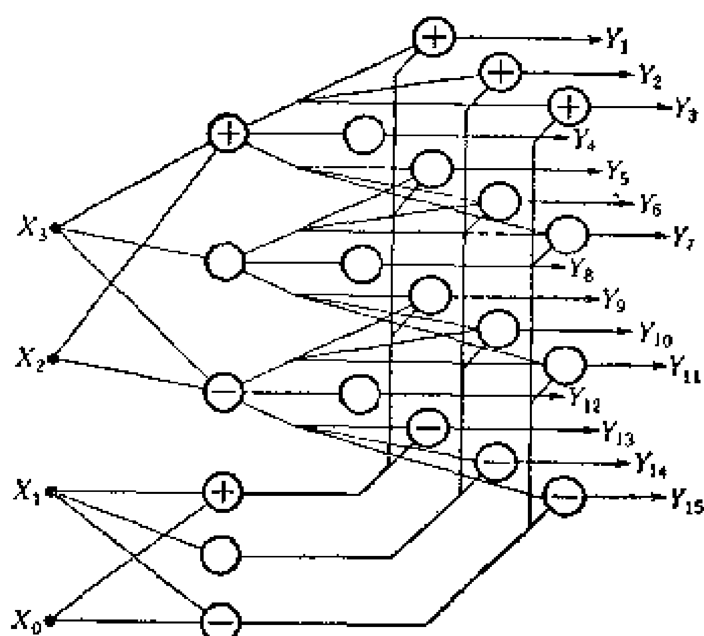


图 25.2 "阶段型"譯碼器

只有一个1，其他都是0，这时根据输入的情况組成 m 个輸出的組合，是与前述譯碼器相反的綫路。

与譯碼器相反，編碼器以邏輯和元件构成，即对一个輸出端，将 N 个輸入中使这个輸出为1的部分集合，以邏輯和綫路混合。图 25.3 是将 2^n 个輸入按二进制編碼輸出的綫路。編碼器有各种使用目的，并不限制輸出端数目 m 一定要小于 N. 但是，它的特征是輸入信号必須"N 中只有一个1".

将譯碼器与編碼器組合起来，若将前者的輸出作为后者的輸入，则得到由 n 个輸入端进来的二进制符号，被 2^n 种符号选擇輸出的装置。称这种机能的綫路为**函数表**，用于解釋計算机的指令，并产生控制各部分的信号；也用于**翻譯符号**。

譯碼器、編碼器可用机械的方法产生（如图 21.2 的缺口机构）。电傳打字机的发号鍵盘，是应用机械的編碼器选擇发信符号，它的印刷机在6位(bit)机械譯碼器的 2^6 个（实际約50个）鉛字杆中选出1个，将文字印刷。同样，电傳打字符号及鉛字杆的排列也

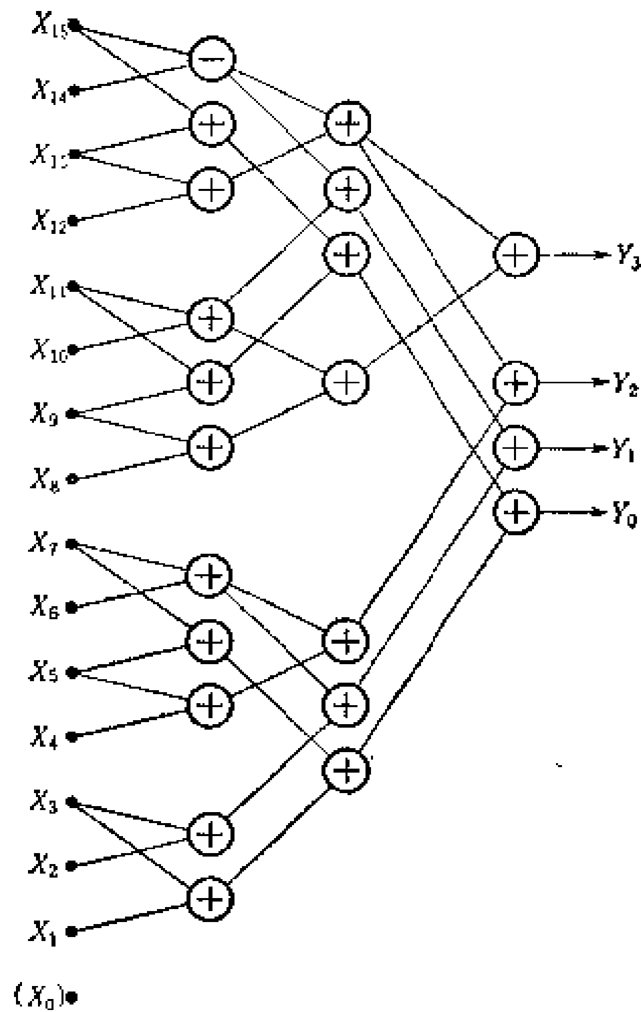


图 25.3 二进制編碼器(4 位)

是利用缺口的編碼条(code bar)以产生譯碼和編碼的作用。

(iv) **同时輸入的計数綫路(檢查綫路)** 即檢查 n 个輸入端中有几个"1"的綫路，其构造示于图 25.4. 它的第 $(n-1)$ 級的輸出

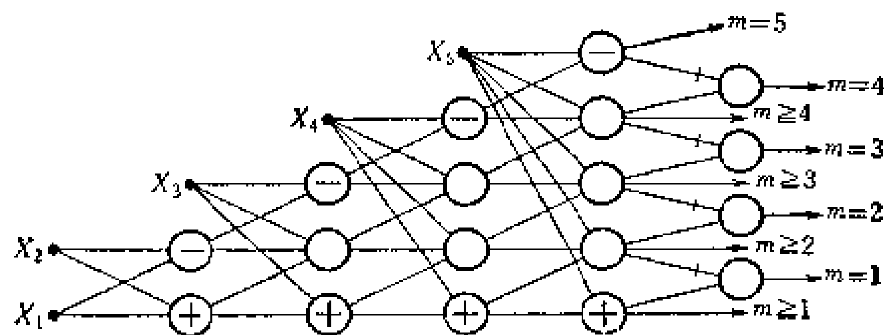


图 25.4 計数綫路(檢查綫路)

指示有 m 个以上的1．为了得到 m 个“1”的指示起見，可以在第 $(n-1)$ 級的相邻元件之間接入比較元件(第 n 級)。

假若輸入是“n 中取 1”的，那末这种綫路实际上成为檢査輸入是否只有一个“1”或是其他情况的綫路。

(v) **运算綫路** 对二进制或編碼的十进制等表示的数进行加減法运算的綫路，是重要的邏輯綫路之一。在 §42 中将会讲到，这里从略。

第2章　有限自动机的理論

§26　自动机和邏輯代数

計算机从效能上讲，它是一个"**暗箱**"，从**輸入端**将以适当方法編碼了的数及計算内容的說明(程序)送入，就从**輸出端**以适当方法編碼了的解答数值等等輸出。这样，不管輸入端有几个，輸出端有几个，輸入端以**离散的**信号送入信息，輸出端也以离散的信号得出輸出信息，这种暗箱就叫做**自动机**(automaton，复数 automata)。計算机是自动机的一种，此外如电話交換机，鉄道信号装置和轉轍机系統等都可作为大規模的自动机。

上述自动机的定义是局限于此的权宜說法。更广泛的，連續輸入及輸出的信号系統也叫自动机，这时也是同样考虑这些因素。另外，不必限制輸入和輸出是怎样的信息，例如轉动把手，投入銅币等作为輸入，輸出是电灯亮，留声机响，彈子或香烟出来皆可。这些变化仅是物理实現的差別，在本质上，数学的、抽象化的实质是相同的。今后只考虑抽象的輸入、輸出信息。

自动机的数学描述，即是規定自动机的"現在的輸出"是怎样的"过去的輸入"的函数。此种自动机是**决定論**(deterministic)的，其表达工具是邏輯函数(即輸入是邏輯变数，輸出是这些变数的邏輯函数)。本书不考察伴随**随机性**(randomness)的非决定論自动机。

邏輯函数由基本"邏輯操作"的形式組合来表現，与之相对应，正如下面所示，自动机可由几个基本的自动机元件的物理組合加以实現。所謂組合，就是把某自动机的輸出端，接至其他(有时是自身)自动机的輸入端，前者的輸出作为后者的輸入。

自动机的元件，随着輸入、輸出的物理形态不同，而有許多种，典型的如 §21, §22 中的机械的缺口元件，继电器，整流器（二极管），电子管，变参元件等，它的組合方法在不同的場合有种种差别。但是，一般都用执行 &，∨ 或相近的基本邏輯操作（Sheffer 的划函数，多数决定操作等）的元件的組合而构成复杂的自动机。因此，邏輯函数的基本操作的分解，是与自动机的基本元件的分解相对应的，过去所述各定理当然可以应用。

但是，不能不看到，邏輯式与其所表現的物理装置之間，有很大的差别。邏輯式的左边及右边是对等的，而继电器，变参元件等元件的輸入与輸出之間，只有单向的因果关系，例如

$$Y \equiv X_1 \cdot X_2 \vee X_1' \cdot X_2' \qquad (26.1)$$

及

$$X_2 \equiv X_1 \cdot Y \vee X_1' \cdot Y', \qquad (26.2)$$

这两个式子是表現同一內容的，而它們所代表的物理装置（計算机）則完全是不同的东西。(26.2) 式是将計算机的輸出端以 X_2 置入，这时要从輸入端获得 Y 是不可能的。邏輯式，或邏輯代数不解决原因与結果的問題，只不过单純地表現实际存在的关系。

邏輯式沒有包含但在物理中存在的因果关系，它的本质是怎样的呢？这是时間上前后的关系。不管速度怎样高，結果总比原因慢❶。要正确地得到自动机的特性，必須在邏輯式中引入时間的因素。

有了这种延迟，就会发生新的情况。在邏輯代数中，

$$X \equiv X$$

是明显而沒有內容的廢話，可是有物理的解釋。就是一个什么运算也不做，只是将它的輸入經过中继而輸出的简单的計算机，把它

❶ 自动机所用的元件，例如电子管延迟 $10^{-6} \sim 10^{-7}$ 秒，继电器延迟約 2×10^{-3} 秒以上，目前用的变参元件延迟約 10^{-5} 秒。

的輸出反饋到輸入端。若这时輸出是 0，則輸入后再作为輸出的还是 0，形成 0 的循环；同样，若这时輸出为 1，則形成 1 的循环。因此，产生了**存貯**机能，所謂**触发电路**（**反复电路**）(flip-flop circuit) 就是一个实际的例子。

另外，

$$X \equiv -X$$

在邏輯代数中是虛假的，也是沒有意义的。但物理上可解釋为仅有一个否定元件构成的計算机，将輸出反饋至輸入。因此某一瞬間的輸出是 0，送至輸入端作为輸入，則以后的輸出是 1，这个 1 又作为輸入，則下面又輸出 0，多次下去就形成 0, 1, 0, 1 **振动**。例如**多諧振蕩器**(multivibrator)及电鈴的振动就是这些例子。若測定此振动的周期，也就是这个“計算机”从輸入到輸出的延迟。

在自动机的理論中，通常把实际自动机元件的各种各样附带的复杂性质除去，加以理想化、抽象化以至数学体系化。

在自动机的理論中，所使用的語言是与实际物理装置对应的术語。邏輯变数总是时間 t 的函数，t 是不連續的，考虑作**离散的**，卽 t 只取**整数值**。

今后考虑的純理論的自动机，对于每个輸入都有一定的**延迟** δ_i. 因此，以前的 $Y=f(X_1, X_2, \cdots, X_n)$ 今后都写成

$$Y(t)=f(X_1(t-\delta_1), X_2(t-\delta_2), \cdots, X_n(t-\delta_n)),$$

δ_i 全部是整数。

这个假定对于实际的元件，有些場合还是不能满足。电子管及继电器等元件輸入至輸出間的延迟决不能明确划定，另外，輸入及輸出从 0 到 1 或从 1 到 0 的变化时刻在时間上也不够明确，有中途的不确定时期。因此，上述的理想化假定对之不甚适用。

然而在有些計算机中，全部机器的动作由一个独立装置的**时标脉冲发生器**产生的信号支配。全部变化是在一个时标脉冲

至下一个时标脉冲的时間中进行的，这种机器称为**同步式机器**(synchronous machine)．对这种情况上述规定可完全适用。变参元件是同步式元件的一种，一个变参元件的輸入与輸出之間有一定的时間延迟存在。

沒有时标脉冲的机器称为**异步式机器**(asynchronous machine)，多数的继电器計算机及其他自动机属于这一类。它們的数学处理与实际的設計比較困难，但很多地方却比較簡省。

因此，上述自动机的理論是与同步式机器相当的，今后主要就变参元件(同步式元件)加以說明。可是，在运用了适当方法之后，对异步式机器，自动机理論也可能适用。

§27 环及路程差

現在考虑将变参元件(邏輯元件)以各种各样的方式結合，构成**变参元件网絡**(parametron net)的情况。从一个变参元件出发，沿着作用傳輸的路徑找寻下去。当然，路徑是有很多分支的，有时再汇合，在这許多路徑中，有时再返回前面的地方(图 27.1)，这时，作用就沿着环路循环。这个网絡，称为有**环**(circle)网絡。

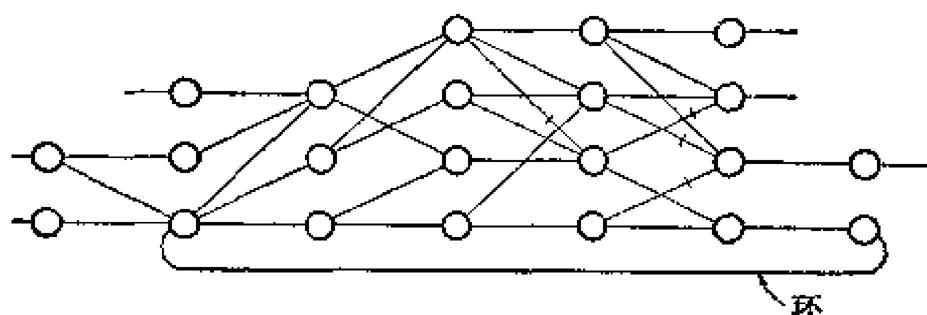


图 27.1 有环网絡

有环网絡无法用单純的邏輯式(不含有 t)表現，虽然式子的左边及右边出現相同的变数，但用同一个方程式是不能得到正确解釋的。在一般情况下，有环綫路的信号是一直循环着的，这时系統产生存貯(記忆)的能力。有存貯能力的系統，不能由系統

的現在状态或輸出，以及此时刻以前的有限时間內的输入来完全确定，而要根据系統的最初（实际装置是电源开关接通时刻）状态，才能加以完全确定。

另外，同样的路綫图上，一度分开，再汇合，而且考虑两方面的路程不相等的場合。輸入在某个瞬間变化后，沿两条通路前进的信号也将发生变化，在汇合时，其中一条綫路的信号已先变化，而另一条綫路还仍象开始时一样。此时，仅仅在变化轉移的瞬間，发生与正常状态相异的特殊状态，且有特殊的輸出产生（图 27.2）。这个有路程差的綫路（图 27.2）可用于产生单脉冲（只在 1 个单位时間內为"1"，以后又立即复归"0"的信号）。

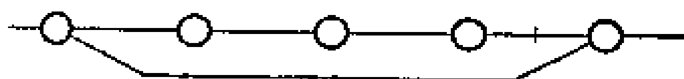


图 27.2 有路程差綫路的例子（微分綫路）

在自动机中，輸入、輸出是作为 t 的函数。当网絡不含环的时候，一般輸出 $Y(t)$ 是输入 $X_1, X_2, \cdots, X_n$ 及 $X_1(t-1), X_1(t-2), \cdots, X_1(t-l); X_2(t-1), \cdots, X_2(t-l); \cdots; X_n(t-1), \cdots, X_n(t-l)$的函数。这里，$l$ 称为自动机的**留效长度**❶，它不长于从輸入至輸出的最长的路程。在这种情况下，若 Y 是上述变数的函数关系已完全确定，則自动机的外部性质（仅仅注意輸入与輸出关系的性质）也完全确定，这就属于邏輯代数所反映的問題范圍。

留效长度为 l 的自动机，可由上述 ln 个参数的邏輯函数表示，它由輸入的 2^{ln} 种組合及其所对应的輸出表格决定。这种自动机 Kleene❷称之为确定性（definite）的自动机。处理确定性自动机时，引入时間因素并不会使問題复杂化。

❶ 关于留效长度的定义参看 §28. ——譯者注

❷ Kleene 为自动机理論的首創者之一，其第一篇关于自动机的經典文献发表于《Project RAND Research Memorandum RM-704》 pp. 101, 1951, 12. 已有中文譯本《自动机研究》，科学出版社 1963. ——譯者注

給定变参元件网絡，若其中有环路，并且将环的長度(信号沿环路傳輸一周所需的时間)与迂回路綫的路程差全部考慮时，这些参数的最大公約数 m 称为网絡的**周期**。

若 $m \geqq 2$ 时，可将网絡的构成元件分成 m 組。在划分时以某一个元件为基准的属于第 0 組，以其輸出为輸入的下一級元件属第 1 組，而第 1 組元件的輸出由下一級第 2 組的元件接受。供給第 0 組元件輸入的，是属于第 $(m-1)$ 組部分的元件。一般說，第 i 組元件的輸出由第 $(i+1)$ 組 (mod m) 的元件接受。按此規則确定元件的划分时，必須不发生同一元件不得不属于两个組的矛盾❶。

将元件所属的部分(組)，称为元件的**相**，如称为第 i 相的元件。

某瞬間 t，在第 i 相元件所保留的信号，在 $t+n$ 时刻，信号必然傳至属于第 $(i+n)$ 相 (mod m) 的元件上。相反，第 i 相元件在 t 时刻的状态，由 $(t-1)$，$(t-2)$，… 时刻属于各第 $(i-1)$，$(i-2)$，… 相 (mod m) 的元件的輸入信号决定。其他相的元件在那个时刻的状态不影响这个元件。

将周期 $m \geqq 2$ 的网絡中傳播的信号自身分解成 m 組，属于不同組的信号，在綫路中运动时，相互間毫无影响。因此，考慮 $m \geqq 2$ 的网絡时，可不必同时考慮全部信号。用通信工程的語言說，就是将 m 組独立信号进行**时間分割** (time division) 或**多重化** (multiplex)，我們只要着眼于一組討論即可。

“現在实用的变参元件綫路，全部有三个周期即可，因此，元件分为三相。”

然而，在上述說明中，并非是三組独立信号的混合运动。实际上，根据变参元件的工作原理，三个之中只有一組信号是实际存在的，其余二組不是实际存在的。而且，在实际变参元件中，只有属

❶ 这个規則是針对多拍綫路的，并非普遍規則。——譯者注

于某一相的元件有活动，其余二相都休止。这是因为从外面向变参元件置入新的信号时，有必要先使它一度休止，以使于前面的信号完全消失，这相当于神經細胞的不反应期 (refractory period)。

我們今后的討論，仅着眼于三組中的一組信号，其他的相在問題中不必使用。所有变参元件网絡，其周期为三。当然，可以使用更大的周期(4，5 等)，但实际上并不应用。

无散逸綫路 周期为 ∞，亦即沒有环，也沒有迂回路綫的网絡，称为**无散逸綫路**。这时，全部元件分为很多相，第 i 相元件輸出是第 $(i+1)$ 相元件的輸入，各相完全順次序排列。某一相的元件，在某一瞬間发出的信号，以明确的相位波前傳播，直到最后一相(也許是輸出元件)为止。这时，第 0 相的元件 P_j 的状态 $X_j(t)$ 仅与 P_k 相关，P_k 是第 i 相的元件，其时状态为 $X_k(t+i)$。这里，仅取相对应的时刻，于是时間的因素消失了，可作为簡单組合的 (combinatoric) 問題处理。

組合的綫路，是輸入变数的邏輯函数，§ 23～§ 25 都是以变参元件实現的例子。

非确定性自动机 有环綫路的情况非常不同，最簡单的是只有一个环的情况(图 27.3)。这时，輸入两端，假如都置以 0，环中的信号是 0 还是 1 并不确定，要看以前在环中循环着什么信号。这种不确定性，Kleene 称为**非确定性自动机** (indefinite)。在非确定性自动机中，某个元件的現在状态 $Y(t)$ 是受无限的过去的輸入影响的，以有限項目构成的表，无法描述此种动作，所以导出了下面的各种各样的問題。

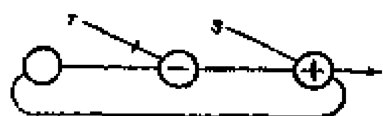


图 27.3 有环綫路的簡单例子 (触发綫路)

§ 28 自动机理論問題的展望

在上述自动机理論中，主要的問題是非确定性自动机。問題

分为两个方面：(i)給定具体的綫路网絡(例如变参元件网絡)，将其动作全部描述出来——特别是，希望将二个具有全同的外部特性的网絡，以同样的标准形式描述下来；以及(ii)給定某个特性，导求实現它的綫路网絡。前者是**分析**問題，后者是**綜合**問題。

处理这些問題，又有两种方法。一种是自动机的**外部行为**(external behavior)，即将輸入和輸出的关系直接列举出来，亦即将产生某个輸出的所有輸入列出的方法。在确定性自动机中，实际是可能的，但在非确定性自动机中，一般輸入系列有无限的可能，用簡单的表是不可能作出的。在这里导入将所有可能系列，以有限形式表示的符号，即 Kleene 的'*'操作(函数)，并依据所謂**正規表現**(regular expression)，則(i)，(ii)两方面的問題大体上得到解决。

另一个方法則是代替輸入及輸出关系的直接考察，而将过去輸入信号在現在的效果(留效)，反映为系統**内部状态**(state)，也就是考察每时刻的輸入信号所引起的系統状态的相应变化(**迁移** transition)，通曉迁移法則(**迁移图**，transition diagram，state diagram)，从而描述系統的行为。輸出根据各瞬間的状态和当时的輸入来确定。此方法是分析实际自动机的有效手段。

§29 迁 移 图

上节中第二种方法，就是从考察自动机的内部状态出发，将自动机的給定构造(例如变参元件綫路)进行分析。

給定周期为 p 的变参元件网絡。构成网絡的全部元件以某种形式联結，这些元件分为三种，即**輸入元件**，**内部元件**和**輸出元件**。輸入元件是直接接受外面的輸入信号的元件，它不从其他元件接受輸入，而仅向它們輸出。若輸入变数有 n 个，則輸入元件有 n 个，輸入元件以小圓表示。内部元件的輸入、輸出綫总是与其他

元件結合的。輸出元件是将輸出信号取出的元件，而从其他元件接受輸入，不向其他元件送出輸出。但輸出元件作为綫路的一个环节，有时需要将信号引导到其他元件，这时再設置一个元件，接受同样的輸入信号，用作内部元件。

在这种綫路中，对属于某一特定相的元件观察，为方便，取0相观察。其中的一个确定元件，在某一动作的瞬間取0状态，还是取1状态，由給出这个元件輸入信号的第$(p-1)$相的元件(1个也是3个)在此瞬間之前的状态决定。而此$(p-1)$相元件的状态是由$(p-2)$相的元件，在前一瞬間之前的状态决定。按着信号流动的相反方向，追寻原来行徑，若有分枝时，某些分枝結束于輸入元件。但可沿着另外的分枝追寻下去，經过p級之后再回到第0相的元件。此处，第0相元件在t时刻的状态，是由一周期前即$(t-p)$时刻的同一相的全部内部元件的状态，以及在$t-p$与t之間的輸入元件的状态所完全确定。当第0相的内部元件有m个时，此自动机的全部内部状态有2^m个或以下；若这些状态和輸入即n个輸入元件的状态已經确定，則此时刻t的n个内部元件的状态也被确定。这个自动机的动作是从某瞬間$(t-p)$到p个間隔以后的t时刻，从一个状态S_1迁移至另一状态S的。这个状态S是由前一状态S_1及迁移途中的輸入元件状态决定的。若2^m个状态以圓圈表示，其間的迁移以带箭头的綫条表示，将輸入組合附記在迁移綫旁边，画成图表，則这种图表将系統的行为完全描述了。图29.1是迁移图的簡单例子，从各个圓射出的綫对应于全部可能的輸入，既不要遺漏也不要重复。反之，进入一个圓的箭头对应于怎样的輸入是不一

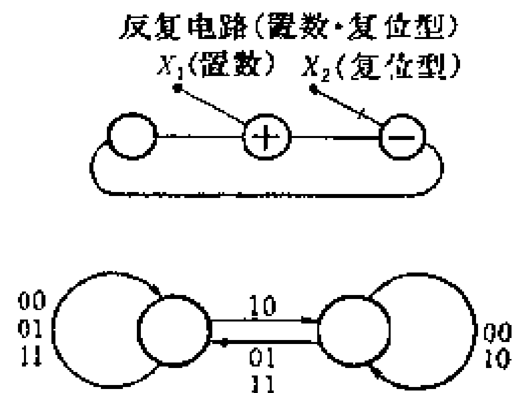


图29.1　簡单迁移图的例子

定的，特别是某些圓連一根輸入綫也沒有，这种状态，从某种意义上說是**过渡状态**。这种状态在系統的最初（开始接电源）阶段有可能存在，但系統立即又离开这种状态，以后也不再回到这个状态。在平常考察系統的行为时，可认为这状态不存在。

对自动机的动作作一般的考察时，对輸入信号 $X_1, X_2, \cdots, X_n$ 不須作本质的限制。总之，輸入信号有 2^n 种，以文字 X 表示輸入，并取 $0\sim2^n-1$ 个整数作为 X 的值。有时，2^n 个組合中的一部分，实际不表現出来，这时，实际表現的輸入只有 N 种可考虑了。

不連結状态　随便取出两个状态 S_i, S_j，若經常存在从 S_i 向 S_j 状态进行的鏈，则此机器称为是**强連結**的。在后面时钟的例子中，迁移图是无分枝的、具有以 2^n 个状态为周期的环，这是强連結的，有分枝的就不是强連結。在不是强連結的机器中，其迁移图常可取出一部分是强連結的。此强連結状态的部分的集合，既不失去上述属性，又不能再行扩張，则称此集合为迁移图的**根**（root）。在迁移图中，同一情况下可以有 n 个根，他們之間或为完全不連結的場合，或为弱連結的場合，后者即为从一个根的状态总可向另一个根的状态过渡，但不可以倒过来进行。这时根就可以排出次序。在迁移图中，根以外的部分称为**枝**。考察只有一个状态形成的根，这是存在自身回到自身的回路的情况。

若一个机器，由完全不連結的 k 个部分組成的迁移图表示，则按照开始时它所具有的状态，可表現出完全无关的 k 种行为。即等于有 k 种不同的机器存在，当令机器属于那一种，則由最初状态完全决定。

在实际机器中，存在不連結状态时，很不方便。这时，在机器中，当最初的开关接通时，需要按指定初始状态的电鈕，以产生强制的輸入状态。因此設計时必須要使全不連結的状态不存在，而

全部状态間至少也有弱連結存在。

§30 基本綫路的例子

（i）**反复电路**（图 30.1） 以三个变参元件构成一个环，寄存一位信号。有置数輸入(s)及复位輸入(r)，在 s 端輸入信号为 1 时，内部状态被强制变为 1，在 r 端輸入信号为 1 时，内部状态被强制变为 0. 当輸入 r, s 均为 0 时，仅存貯原来信息。图 30.1 的綫路中，r, s 双方同时加入 1 时，若 r 强于 s，则状态为 0. 反之，s 强时，則后边向 s 的方面轉变，通常不允許有双方同时輸入的情况出現。

在图中的(b)也是一种反复电路，当 s, r 同时輸入时，有抵消作用，与輸入 0 是相同的。

图中的(c)是在**一位寄存器**中的一种反复电路，从信号輸入端輸入的信息只有在另一門輸入端是 1 时才能"写入"回路。門輸入端是 0 时回路只存儲原来信息。

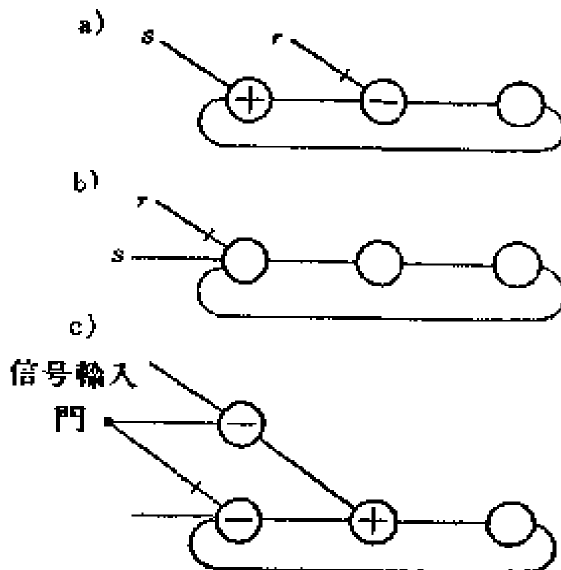


图 30.1 反复电路

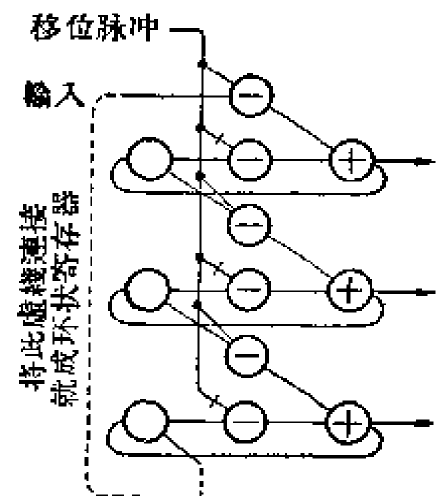


图 30.2 移位寄存器

（ii）**移位寄存器** 将图 30.1（c）一位寄存器連成 n 个鏈状，一个寄存器的内容成为下一級寄存器的輸入信号，并給与共通的

門輸入信号，則构成图 30.2 的綫路。在加上門輸入信号时，信号沿着一个环向下一环傳送移动，同时从第一端新信号又进入，一个接一个地将 n 位輸入信号寄存在 n 个环中。这种将一端順序傳送的信号轉变为靜止的寄存信号的綫路，称为移位寄存器 (shift register)，使控制信号移动的門信号称为**移位脉冲**(shift pulse)。其用途是将逐次的串行信号組成同时并列的信号 (staticiser)；或相反，将同时輸入的并行信号，向一端逐次串行輸出 (dynamiciser)。这是計算机及其他装置用于寄存信号的基本綫路之一。

若从移位寄存器的最后一級連結至信号輸入端，則成为 n 个脉冲循环的**环状寄存器**。其中寄存的信号，在 n 个移位脉冲作用下，可多次取得逐次的串行信号。

特別是当寄存器的置入信号为100…0，即只有一个1，其他全是0时，移位脉冲使这个1逐級移动 n 个脉冲后又回到原位。移位寄存器的这种使用方法称为**环状計数器** (ring counter)，用于向 n 个場所順次傳送信号的場合（同样的目的用于下面的二进計数器及 § 25 的譯碼器）。

虽然在移位信号之外沒有其他輸入时，环状寄存器有 2^n 个不同的状态，但是各状态經过 n 个移位脉冲(或 n 的約数)后即走一周期，因此迁移图由許多互不連絡的环組成。根据最初輸入的模样(pattern)，即可决定属于迁移图的哪个环中。

从最后一級以否定信号向回傳送，就构成**反轉环状寄存器**，它的状态要 $2n$ 个移位成为一周。

(iii) **二进計数器** 将图 23.1，II_2 的二变数邏輯綫路的輸出轉接至輸入，当輸入为1时，得到内部状态从0到1，或从1到0的綫路，称为二进計数器 (binary counter)。这綫路的第1級邏輯积处，当信号从1变到0时，就产生輸出信号。这个信号作为进位信号，接至別的二进計数器的輸入，形成2位的二进計数器。同

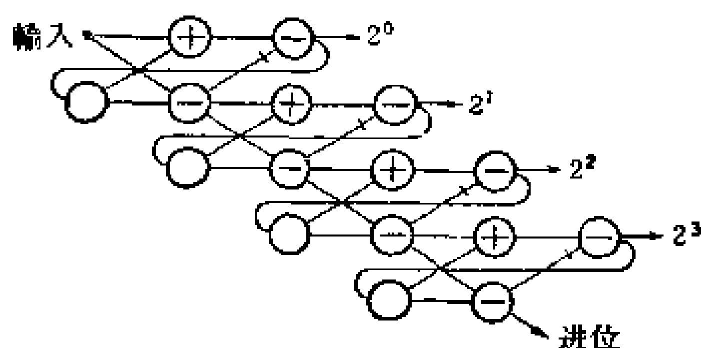


图 30.3　二进計数器(四級)

样,可得 n 位二进計数器(又称为 2^n 个刻度,scale of 2^n)(图 30.3)。将各級寄存的信号排列,則形成計数(count)的二进制表示。計数从 2^n-1(二进制表示中的 $\overbrace{1\,1\,1\cdots1}^{n}$)回到 0 时,从最后級送出进位信号,成为每輸入 2^n 个"1"給出一个輸出的装置。

将最后級进位信号送回到适当的中間級(反饋),最初就不从 0 开始,而从一个予置(preset)数 m 开始,則形成一个 2^n-m 为一周的計数器。这里可不以 2 的冪数作計数的进位輸出,例如**十进計数器**(图 30.4)的結构。(这里是进位先出现的方式,其他还有各种各样的方式。)

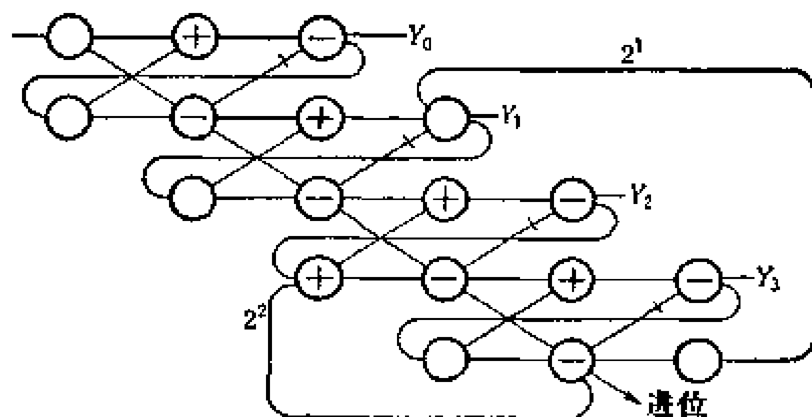


图 30.4　十进計数器(予置 6=0110)

§31 时　　钟

今考虑一个輸入也沒有的自动机,称之謂**时钟**。普通时钟就

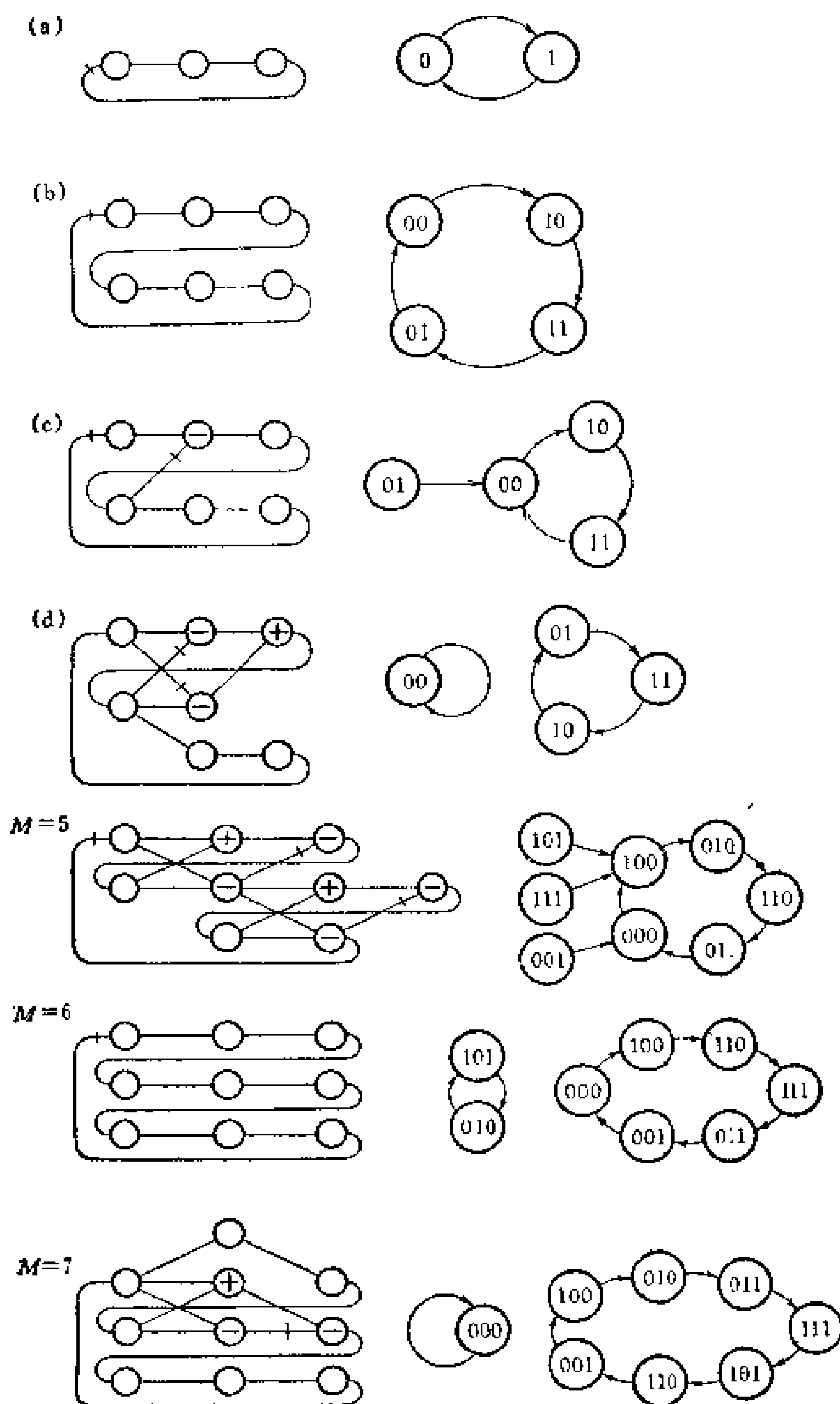


图 31.1 "时钟"的綫路及其迁移图的例子

是沒有輸入的自动机（卷发条与对时不算）。“停表”則不属此类型。时钟对于串行計算机和周期控制的各种机器是必需的。

时钟的迁移图，只有一种状态迁移方式，它必然是 M 个状态的环状形式，有时也分出若干分枝。分枝就是前述的过渡状态。M 称为这个时钟的周期。

图 31.1 是几种时钟的綫路。

§32 状态的等价性，迁移图的簡化

輸出 至今，仅仅討論了自动机內部状态随輸入而变化的規則，借以揭示实际机器內部的情况，若我們从机器外部观察，則得知某些特定元件即輸出元件的状态。若輸出元件在第 r 相，它的状态照例由时間上比 r 更前的第 0 相的內部元件的状态，及从 0 相至 $r-1$ 相間的輸入元件的状态决定。因此，可以說系統是“在 S_i 状态由于輸入 X 而輸出 Y，并向 S_j 状态迁移。”系統在一个周期內的动作过程中（Y 仍然是几个輸出元件的状态的綜合），各 j，Y 都是由 i，X 唯一地确定的。

迁移图的簡化 至今所表示的迁移图，是对机器內部調查后，将其状态的变化示于图上。但是，即使不看机器的內部，而只看輸出，或观察以后的輸入与輸出的关系，仍然有必要知道在这个瞬間机器內部有几个內部状态。即輸出受到几个周期以前的輸入的影响形式如何，此影响存貯在机器之中，其存貯內容即以內部状态表示。

但是，在这种意义上的內部状态与前述的眞正的“內部”状态之間，不限于 1 对 1 地对应。若从观察二个輸出之中，判断两个状态确实不同，則眞正的內部状态也确实是不同的。若內部状态相同，輸入也相同，則形成相同的輸出。此处，这种对应仅对一个方向是唯一确定的。“仅仅調查輸入和輸出的关系而不能加以区别

的两个状态"称为**相等**。

图 32.1(a) 的綫路中，由于元件 P_1, P_2 各有两个状态，内部状态共有 4 种。图(b)为其迁移图。但是，有区别的(独立的)只有 3 种，即 S_{11} 是当輸入为 1 时，輸出为 1；S_{01} 是輸入为 1 时輸出为 0，并移向状态 S_{11}. 但是，状态 S_{00}, S_{10} 沒有区别，因为两者都是当輸入为 1 时向 S_{01} 移动，得到輸出 0，而当輸入为 0 时向 S_{00} 移动。所以得到三个状态的簡化迁移图(c)．这个机器是檢查"三次連續来 1"的机器。三个状态的作用是：現在每次来 1，綫路都把它存儲起来，来过三个 1 以后，每次来 1 时机器都輸出 1.

所謂两个"状态"相等，即是各有这状态的两个全同的机器，給定相同的輸入系列，观察輸出的各个状态，常有相同的輸出系

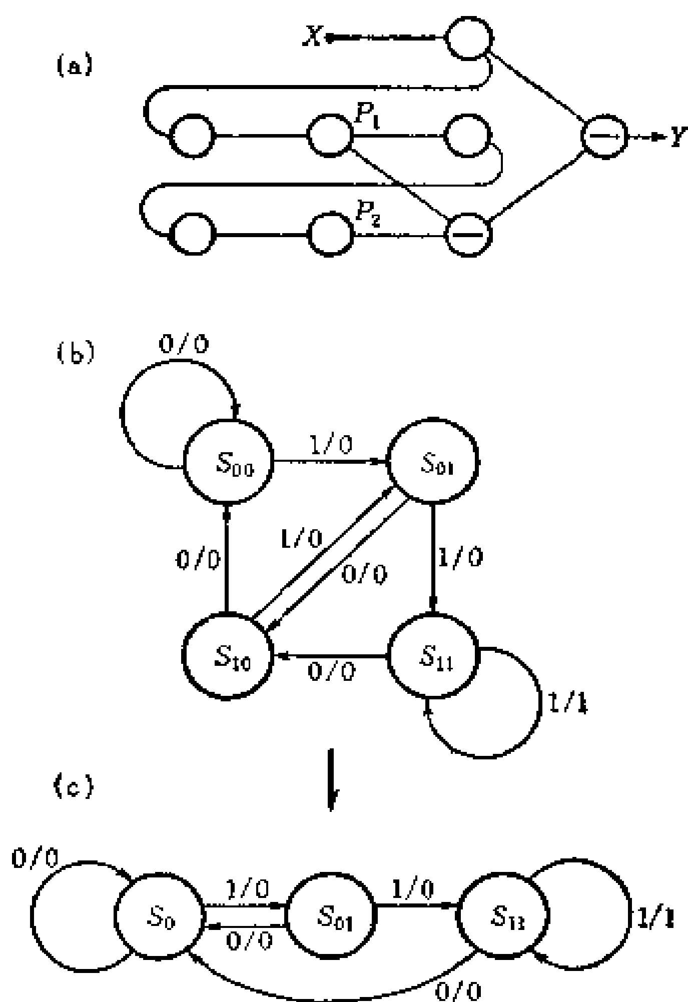


图 32.1 迁移图的簡化

列。总之，对于机器的"輸出方面的信息"，即使对此两状态不加区别也毫无妨碍。

两个状态是否相等，实际可从各状态出发，将各种各样的输入系列加于两方面，观察是否有不同的輸出表現。若状态不同終究要产生不同的輸出，但究竟需要試驗几次才能发現"一个"不同的輸出？有如下的定理对照。

定理 32.1 内部状态数为 M 的自动机，假如有两个内部状态不同，一定存在长度为 $(M-1)$ 个以下的輸入系列，以区别这两个状态。总之，将 $(M-1)$ 个輸入的所有可能組合（N^{M-1} 种）的各种状态，加于某（两个）自动机，两者輸出常相等，則两状态是相等的。

在証明时，首先从各状态出发，給定各种輸入系列，将輸出比較考察。两状态若不相等，必存在一个整数 k，且一个长度为 k 的系列能将两者区别出来，而 $(k-1)$ 长度的系列一定区别不出来。依据这个长度为 k 的特别系列，历訪所有的状态的对偶（两个机器），不会出現二次全同的对偶。假如有相同的对偶出現，則因为可加以省略，所以有一个比 k 短的系列将两状态区别。而且，在訪問第 $k-h$ 号状态的对偶时❶，加上长度为 h 的輸入系列就能給以区别。所以，对于在 $1 \leqq h < k$ 中的全部 h，至少有一对状态，被恰好是长度为 h 的輸入系列所区别。

那末，是 $k \leqq M-1$ 还是 $k > M-1$ 呢？若 $k \leqq M-1$，則定理成立。今假定 $k > M-1$。

現在加一次輸入，所得的輸出的 M 个状态至少可分为二組。再加一次輸入，根据輸出又可对状态更細地分类。同样，对輸入系列的长度根据状态也可分为很多組。系列长度每增长 1 則至少增加一組数，因此必存在长度为 $h(1 \leqq h < k)$ 的系列，将状态的对偶区别。如此，在长度为 $M-1$ 以下的系列中，就能将 M 組状态分类，

❶ 輸入系列是按序排列的，故可从小到大地編号。——譯者注

总之全部状态分为不同的組，所以与 $k>M-1$ 的假定矛盾。

(証毕)

此定理的結論中所涉及的值 $M-1$，不会再減少。实际，可以用 $M-1$ 的系列产生区别的例子来論証这一結論。例如 §31 的时钟，仅有一个状态(S_0)产生輸出 1，其他輸出皆为 0，而輸出为 1 的下一个状态 S_1 及再下一个状态 S_2，直到第($M-1$)号的 S_{M-1} 与 S_0 到来的前一瞬間才看出区别❶。

这个定理至少从原理上給出区别迁移图状态相等与否的方法。这里，在画出迁移图时，一律将相等的状态作为一个状态。由相等状态出来的相应于同輸入的綫，其目的地也是相等状态(若不如此，则相等的意义就相反了)❷。因此，仅显示不恒等状态的迁移图，称为**简化的迁移图**。

§33 操作表

被简化的迁移图的意义之一，是能表示自动机的**外显性质**，即輸入輸出关系的一个标准型式。給定被简化的迁移图，不仅規定了自动机的外显性质，而且当二个自动机的外显性质全同时，则被简化的迁移图也具有相同的形状。因为，被简化的迁移图所示的状态，不包括机器的內部特殊状态，只表示决定机器未来輸出的实际必要的过去信息，所以可只从机器的外显性质决定。

更直接的表示方法是自动机的**操作表**。所謂长度为 k 的操作表，是从某特定状态出发，給定任意的輸入系列 $X(1), X(2), \cdots, X(k)$，得輸出系列 $Y(1), Y(2), \cdots, Y(k)$，而对所有輸入系列列出的表。对应于各状态(起点)有各个操作表的表示，从状态 S_i 出发，长度为 k 的操作表記作 $\mathfrak{T}_i(k)$，若定理 32.1 中的 S_i 与 $S_{i'}$ 不相

❶ 这里共有 $M-1$ 个状态是沒有区别的。——譯者注

❷ 参看图 32.1(b)，(c)。——譯者注

等，則

$$\mathfrak{T}_i(M-1) \neq \mathfrak{T}_{i'}(M-1),$$

此处，$\mathfrak{T}_i(M-1)$ 是以 S_i 为特征的。在簡化的迁移图的情况下，S_i 和 $\mathfrak{T}_i(M-1)$ 是一一对应的。

从 $\mathfrak{T}_i(M)$ 中除去最初的一行 $Y(1)$，留下部分表示对应于从 S_i 跨越一步而达到的 S_{i1}, S_{i2}, … 等状态的表，即 $\mathfrak{T}_{i1}(M-1)$, $\mathfrak{T}_{i2}(M-1)$, … 的集合。假如已知 $\mathfrak{T}_i(M)$，則可知道 $\mathfrak{T}_{i1}(M-1)$, $\mathfrak{T}_{i2}(M-1)$, … 等。并可知道，从 S_i 开始，由某一个輸入将到达哪一个状态？若知道对于所有的状态的次一状态，則根据这些可构成（簡化的）迁移图。因此，若知道操作表，即可构成迁移图。

但是，若考虑到操作表的建立是以不帶有与状态有关的知識为原則的，則分别建立有关各状态的操作表一事是不妥当的。然而，实际上，普通是从机器的一定的**初始状态**(initial state)出发考虑的，即仅考虑从初始状态 S_0 出发的操作表 $\mathfrak{T}_0(k)$. 此时，下面的定理成立。

定理 33.1 根据从特定的状态出发的长度 $2M-1$ 的操作表 $\mathfrak{T}_0(2M-1)$，可将强連結的自动机的外显性质完全确定。

因为，在强連結的自动机中，从 S_0 出发，可在 M 步以内达到所有的状态。即在强連結条件下，无論如何可在有限步驟內达到所有状态，其所經路綫的全体，沒有环路及一度分枝再汇合等情况，汇合处仅有单純的分枝（假如有的話）連結。因为此路綫上状态的总数是 M，所以要到达一个远的状态的距离（包括 S_0）不会比 M 大。

在此以 M 步达到所有的状态后，再給以后 $M-1$ 步的操作表，則上述机器的动作就完全了解了。

此定理所涉及的数 $2M-1$ 不可能再小下去。表示恰好需要 $2M-1$ 步的例子如下：

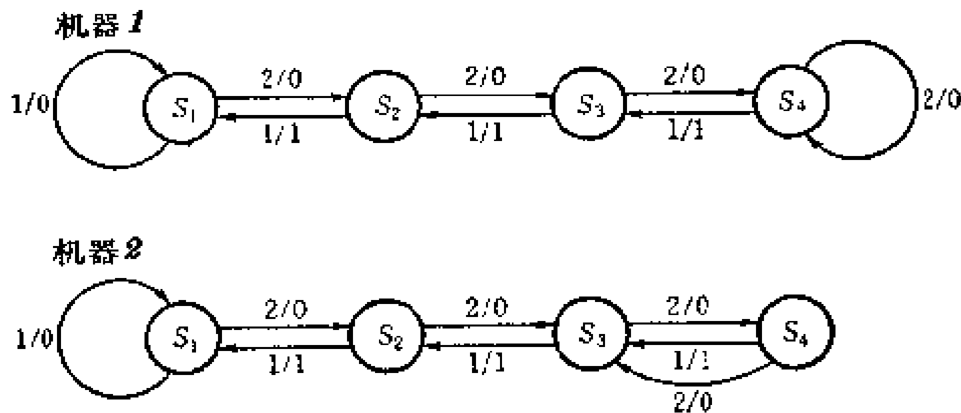


图 33.1　以 $(2M-1)$ 个 $(M=4)$ 輸出产生区别的二个机器

設 $N=2$, $S=2$,考虑下面两机器。

机器 *1*　輸入 1 则 $S_i \to S_{i-1}$,　輸出为 $1(i=2, 3, \cdots, M)$,

　　　　　　　　$S_1 \to S_1$,　輸出为 0.

　　　　輸入 2 则 $S_i \to S_{i+1}$,　輸出为 $0(i=1, 2, \cdots, M-1)$,

　　　　　　　　$S_M \to S_M$,　輸出为 0.

机器 *2*　輸入 1 则 $S_i \to S_{i-1}$,　輸出为 $1(i=2, 3, \cdots, M)$,

　　　　　　　　$S_1 \to S_1$,　輸出为 0.

　　　　輸入 2 则 $S_i \to S_{i+1}$,　輸出为 $0(i=1, 2, \cdots, M-1)$,

　　　　　　　　$S_M \to S_{M-1}$, 輸出为 0.

这二个机器，在 S_M 以外的状态是全同的，所以不达到 S_M 的系列是不会产生区别的。假定 S_1 是初始状态，$M-1$ 步的輸入 2 是达到 S_M 的最短长度。为了区别两者的輸出表現，必須在 S_1 状态下加入輸入"1"，而且在达到 S_1 之前，为了使其不同时起見，在 S_M 状态必須輸入"2"以产生两者的差别。輸入系列:

$$\underbrace{22\cdots2}_{M}\quad\underbrace{11\cdots1}_{M-1}$$

是用以使两者产生区别的最短系列輸入，其輸出各为

$$\underbrace{00\cdots00}_{M}\quad\underbrace{11\cdots1}_{M-1}\qquad(\text{机器 } 1)$$

及

$$\underbrace{00\cdots00}_{M}\quad\underbrace{11\cdots1}_{M-2}0\qquad(\text{机器 } 2)$$

机器 *1* 是自动售貨机的模型，即将硬币投入窗口，并按把手，就跳出貨物(例如車票)的机器。輸入 2 是表示"硬币投入"，輸出 0 表示什么也不給出，

輸出1表示貨物送出。一面将硬币投入，一枚枚硬币在其中积累，表示状态 S_2, S_3, …，以 S_M 代表 $M-1$ 枚。为了不使以后仍然积累起見，接下去状态仍为 S_M. 若达到 S_M 以后，按把手，貨物就出来，而硬币落入机內。若硬币已全部落入后，再按把手，則什么也沒有得出来。（实际机器上还有个"什么也沒有"的輸入，即"0". 这对应于 $S_i \to S_i$，輸出为 0.）

习題 画出出售 15 元一張車票的自动售貨机的迁移图。但是，应滿足下述条件：

1. 投入的貨币有5元和10元两种(入口是一个)。
2. 若恰好收到必需的金額时，則車票自动跳出(不必按把手)。
3. 若有多余的金額进来，应将找零(5元)貨币和車票一起送出。

§34 确定性和非确定性

設自动机在 $t=t_0$❶ 时的内部状态为 S. 在 t_0 以后，給定輸入 $X(t)$，得輸出为 $Y(t)$. 此时，一般的 $Y(t)$ 是依存于初始状态 S 的。若 t 在某值 t_1 以后，即 $t>t_1$，不管 $X(t)$ 怎样，都可以得到与 S 无关的 $Y(t)$，这个自动机称为**确定性**的。若不存在这样的 t_1，即不管 t 有多大，总存在依存于 S 的对应于 $X(t)$ 的 $Y(t)$，这个自动机称为**非确定性**的。在确定性的情况下，有保持上述性质的最小的 t_1，是相对于 t_0 計算的。因为，若給定 $t=t_0$ 时的初始条件，則其后系統的动作仅由以 t_0 为基准的輸入系列决定，与絕对时間无关。此时，对应于最小的 t_1 的 $t_1-t_0=l$，称为这个自动机的**留效长**。

不含有环結构的自动机都是确定性的。因为此时，某一瞬間各元件所持有的信号，总是向一个方向移动，迟早要消失。但是，也有看来是环結构的綫路而实际上信息已經送出完了，而成为确定性的。那么，怎样来判断某个机器是不是确定性的，以及留效长度为多少呢？

---

❶ 今后，对于周期为 p 的回路，取一个周期为其时間单位。所以，变参元件的信号傳輸时間为 $1/p$.

給定机器的迁移图，从某状态出发，若研究其在开始 l 步所到达的輸出，則可以具体知道这机器是否具有留效长度为 l 以下的确定性。但是，留效长度很长的确定性的机器，与真正的非确定性机器則无法判别。至少要研究几步才能确实判断仍然是个問題。这个問題有下述定理成立。

定理 34.1 若內部状态数 M 的自动机是确定性的，則从任意二个状态出发，加入长度为 $M-1$ 的同样輸入系列，則两者必定迁移到相同的状态。并且，这个自动机的留效长不大于 $M-1$.

証明与定理 32.1 的証明相似。从任意二状态 S_i, S_j 出发，順序地加輸入 $X(1)$, $X(2)$, $\cdots$, $X(k)$，設所得状态为 $S_i^{(h)}$, $S_j^{(h)}$. 若机器是确定性的，那末存在 k 值，在 $h<k$ 时，选擇适当的輸入系列及 S_i, S_j，可使

$$S_i^{(h)} \neq S_j^{(h)},$$

当 $h \geqq k$ 时，无論取怎样的輸入系列及 S_i, S_j，一定是

$$S_i^{(h)} = S_j^{(h)}.$$

这样的 k 值卽是留效长，只有 $h=k$ 时才开始

$$S_i^{(h)} = S_j^{(h)}.$$

选擇初始状态 S_i, S_j 及輸入系列，机器所經历的各状态对偶 $(S_i^{(h)}, S_j^{(h)})$ 中，不存在全部相等的对偶。因为，假如

$$S_i^{(h)} = S_i^{(h')}, \quad S_j^{(h)} = S_j^{(h')} \quad (h<h'),$$

这时周期地反复送入輸入系列 $X(h)$, $X(h+1)$, $\cdots$, $X(h'-1)$，两方面状态总是不一致的。这与假設相反。若这里所表示的状态对偶 $(S_i^{(h)}, S_j^{(h)})$ 卽以此为出发点考慮，則恰好是以 $k-h$ 长的輸入系列使状态的区別消失的"状态对偶"。这种性质的状态对偶，在所有 $0 \leqq h<k$ 的 h 中，至少有一个是存在的。

这里 $k>M-1$ 或是 $k \leqq M-1$. 設 $k \leqq M-1$，則与定理一致，因此沒有問題。設 $k>M-1$. 这时与定理 32.1 的証明相反，从

小的 h 方面考虑。首先以长度为 $k-1$ 的输入系列加入，至少有一对状态 S_i, S_j 对应于

$$S_i^{(k-1)} \neq S_j^{(k-1)},$$

在 M 个状态中，以 $k-1$ 长度的输入系列加入后，其状态至少可分成二組（等或不等）。更进一步，以长度为 $(k-2)$，$(k-3)$，… 的输入系列加入后，将其状态按类地细分，每次至少增加一組，直至长度为 0，至少能分成 $(k+1)$ 組。然而，因为状态的总数是 M，必须

$$k+1 \leq M.$$

这与 $k>M-1$ 的假設相反。因此，$k>M-1$ 是不可能的。所以定理 34.1 得証。

所以，对迁移图的所有状态，以长为 $(M-1)$ 的同样输入系列加入，如果总是达到相同的状态，则机器是确定性的；即使只有一个状态不同，则机器就是非确定性的。在确定性的情况下，留效长 k 也許比 $M-1$ 小。如果以长度等于留效长 k 的输入系列加入，而能与初始状态无关地规定一个状态，则此输入系列可作为此状态的特征。所以，常常存在好几个长度为 k 的输入系列，使得导至相同的状态。因此，由所有长度为 k 的输入組合共 N^k 个組成的集合，可划分成若干部分集合，与各个状态一一对应。

在这种情况下，从任意的初始状态出发，只要有长度为 k 的操作表，就能将机器的性能全部描述。此时，直至第 $(k-1)$ 行内，随选擇初始状态的方法而有变化。最后，即 $X(k)$，$Y(k)$ 的栏内与初始状态无关，但是，若 $Y(k)$ 被所有的输入系列所规定，则此机器可用上述系列作为特征。因此，状态总数为 M 的确定性的机器，可完全由长度为 $M-1$ 的操作表作为特征。

上述定理的值 $M-1$，不可能再小，是最优值。由下述例子即知（图 34.1）.

輸入 0 則 $S_i \to S_1$ $(i=1, 2, \cdots, M)$, 輸出为 0

輸入 1 則 $S_i \to S_{i+1}$ $(i=1, 2, \cdots, M-1)$, 輸出为 0

$S_M \to S_M$ 輸出为 1.

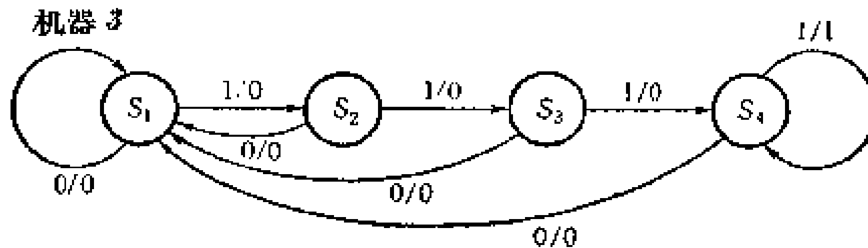


图 34.1 留效长为 $M-1$ 的机器的例子 (将 1 1 1 1 檢查出来)

这是个当連續 M 个 1 輸入后才輸出 1 的机器。S_i 表示連續有 $(i-1)$ 个 1, 因此与以前的状态无关, 显然这是确定性机器。S_1 和 S_2 的区别, 在 $(M-2)$ 个 1 进入时还存在 (S_{M-1} 和 S_M), 但是, $M-1$ 个 1 进入时, 区别就不存在了。因此, $M-1$ 是必要的。

习題 1 画出用来檢查出輸入系列中 0110 系列并輸出 1 的自动机的迁移图(即发見恰好是二个 1 的机器)。

习題 2 图 34.2 是电傳打字机的起止机构 (start-stop mechanism) 的迁移图。对此机器加以各种輸入, 考察其动作。并决定这个机器是确定性的还是非确定性的。

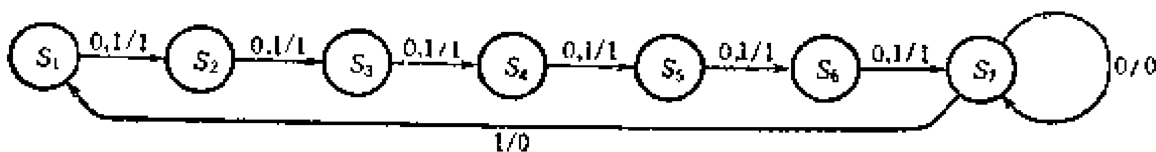


图 34.2 起止机构的迁移图

§35 事件的正規表現

对于描述自动机的方法, 我們已知的有**迁移图**和**操作表**。特别对确定性的自动机, 后者是很直接的方法, 而在非确定性的情况下, 以长度为 $2M-1$ 的操作表可将自动机的性质完全描述, 然而是稍微間接的方法。从操作表上通过任意的(更长的)輸入系列以求得其对应的輸出, 結果即可求得迁移图。但是見到操作表并不

能直接知道“这是个怎样的机器”。此处，更为直观地描述非确定性自动机的性质的方法，即所謂**正規表現**(regular expression)的方法。

正規表現的思想，与过去的都相反，以“回顾过去”的方法作为出发点。以前都是給定輸入系列，考察每次輸出怎样。如今相反，着眼于“某个輸出产生了，过去的輸入是怎样的情况”。在确定性的机器中，若某瞬时 t 的輸出为 1，則可以說出由 $t-k$ 到 t 的 $k+1$ 次輸入系列的所属集合。在非确定性的机器中，不論 k 多大，現在的輸出总是随初始状态而变化的。对于这种无限长的系列，或从 $t=1$ 至 t 时刻的长度变化的系列，要构成完全的操作表是不可能的，因此以某种有限的形式，去表示无限的，或可变长的系列，是正規表現的目的。

与“現在”(即 $t=s$)以前的輸入信号 $X(1), X(2), \cdots, X(s)$ 有关的各种性质，称为**事件**(event)，即輸入信号系列的全部构成的集合(N^s 种)的部分集合。某輸入系列，若确实属于上述部分集合，則事件在現在($t=s$)发生，若不属于部分集合，則事件不发生。另外，一个事件可不仅对特定的 s，而对所有的 s，按照上述的定义加以考慮。

以 $E(s)$ 表示 $t=s$ 时表現某一事件的邏輯变数，当然，$E(s)$ 是作为过去所有的 $X_i(t)$ ($i=1, 2, \cdots, n$; $t=1, 2, \cdots, s$) 的邏輯函数，若 s 变，則它就跟着变。特别在 $E(s)$ 只是 nl 个变数 $X(s)$, $X(s-1), \cdots, X(s-l+1)$ 的邏輯函数，并且用和 s 无关的一定的邏輯式表达时，此事件称为**长度为 l 的确定性事件**。有时，留效长为 $(l-1)$ 的确定性自动机，在 $t=s$ 时，其輸出是 1，或者留效长为 l 的确定性自动机，在 $t=s$ 时有状态 S_i 等等，这些事件确是以长度为 l 的确定性事件。其逆，

“給定长 l 的确定性事件，将此事件表示为 $t=s+\delta/p$ 时的輸

出元件的状态，并以变参元件的綫路构成。δ 是根据事件的邏輯函数的复杂程度而規定的正整数。”

在图 35.1 上从延迟綫电路中取出每隔 p 个号碼的輸出，并以此作为实現邏輯式 $E(s)$ 的无散逸綫路的輸入 $E(s)$，而輸出有某一延迟 δ，以此表示上述事件是明显的。

但是，在 $s<l$ 时，$E(s)$ 中变数在 $t\leqq 0$ 时就变成沒有意义的了。当然，在实际綫路中，照例要有初始状态，其輸出是 0 或是 1，是被明确地規定的。例如，图 35.1 的綫路的初始状态，就是延迟綫中全都是“0”状态，因此在表达 $E(s)$ 的邏輯式中，对于 $s<l$ 的 $E(s)$，特别給以 $t\leqq 0$ 中的 $X_i(t)=0$ 的約束条件。但是，这里特别約定 $s<l$ 时的 $E(s)$ 是很笨的方法，在实际綫路构造时，不必如此。只須以一个一般性的約束条件代替，也就是形式的，无意义的 $s<l$ 时 $E(s)=0$，即此事件不发生。

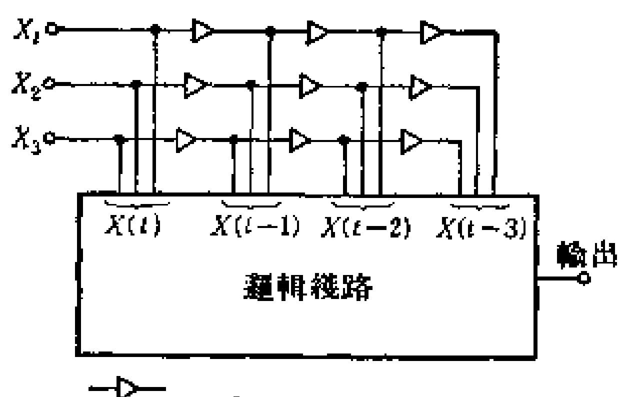


图 35.1　表示长度为 l 的确定性事件

图 35.1 的綫路并不是一般性的，将这綫路跟約束条件統一起来后改成如图 35.2 的，图中有一条长度为 $l-1$ 的延迟綫，最初进入的都是 0，机器动作以后，此延迟綫常輸入 1．将此輸入认为是 X_0，则

$$E(s)=E_1(s)\cdot X_0(s-l+1),$$

即 X_0 經过延迟时間为 $(l-1)$ 的延迟綫的輸出，跟 $E_1(s)$ 取邏輯积，則其輸出满足上述条件。延迟綫右端只在相当于 $t=l$ 时才开始輸

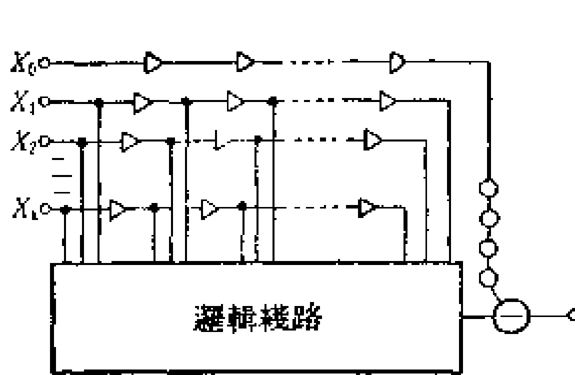


图 35.2 表示确定性事件的綫路

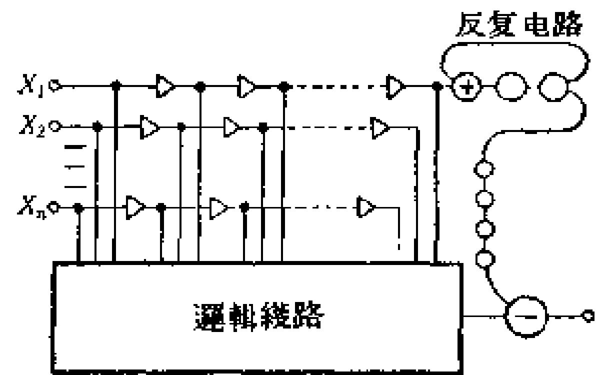


图 35.3 表示确定性事件的綫路

出 1. 另一个方法是图 35.3. 图中有一条延迟綫仅比其他的增长一級;这一級以反饋形成反复电路。初始状态仅仅第一級为 1, 其他为 0. 这个 1 在动作开始后向右前进, 恰好当 $t=l$ 时达到反复电路并将它变成 1. 其后, 保持此状态, 反复电路的輸出与 $E_1(s)$ 的輸出构成邏輯积, 因此也使条件满足。这种方法使用元件的数目就少。

长 l 的确定性事件,若从正常状态($s>l'$)的性质看,也可以看成是长 $l'(l'>l)$ 的事件,但前者在 $s=l$ 时发生的可能性是肯定的,而后者在 $s<l'$ 时决不发生,这是不同的地方。因此,尽管 $E(s)$ 的"长度" l 实际上跟輸出是否依存于 $X(s-l+1)$ 无关,但必須确定。

初期事件 上述长度为 l 的事件是从 $s=l$ 出发,不論何时起皆可,即意味着通常与特定的时間原点无关。但作为一种事件,总是从机器开始动作經过一定的时間才发生的。此时若 l 仍为事件的长度,这时事件是仅发生于 $s=l$, 在以后不再发生。一般是 $X(l)$, $X(l-1)$, $\cdots$, $X(1)$ 的邏輯函数。将此种事件称为**初期事件**(initial event), 在上角記以 i, 如 E^i, 以与非初期事件区别。

以初期事件作为輸出的綫路可构成如下:将图 35.2 的 X_0 再延迟一級,并构成

$$E^i=E_1(s)\cdot X_0(s-l+1)\cdot X_0'\cdot(s-l).$$

同样,也可修改图 35.3 而得。

以长度为 l 的确定性事件作为**单位**，跟邏輯函数的三种基本操作互相組合相似，也可以将它按照若干結合規則而組合成一般的事件的表达式。結合規則根据 Kleene 有如下三种:

(i) $E \vee F$.

这和普通的邏輯和"E 或 F"是相同的,即两个事件,不論那一个发生都形成此事件($E \vee F$)，即使 E 和 F 的长度不同也可。此时,終点($t=s$)是一致的。同样，二个以上的事件以 $\vee$ 結合仍作同样的規定。

(ii) EF.

这与普通的邏輯积不同之处是"E 在 F 以后发生",即将 E 的长度认为是 l,邏輯积意味着

$$E(s)\cdot F(s-l).$$

显然，右方的单位即使是初期事件也沒有关系，而 E 不得为初期事件,这是很明显的。

假如 E 是(i)中所述的不同长度的成分的邏輯和时,一般定义**分配法则**如下:

$$(E \vee F)G = EG \vee FG,$$

则当 E 的长度为 l, F 的长度为 l' 时,上式有

$$E(s)\cdot G(s-l) \vee F(s)\cdot G(s-l')$$

的意义。同样,也可对长度不同的多个事件定义。

下面的分配法则显然也成立:

$$E(F \vee G) = EF \vee EG.$$

乘法的定义中,E 的长度将成为問題,例如 $E(s)$与 $X(s-l+1)$ 无关时，必須区别 E 的长度是 l 还是 $l-1$. $E \vee F$, EF 的演算都滿足結合法則。

此外，E^0F 是 F, E^1 是 E, E^2 是 EE, E^3 是 EEE, …, 一

般 $E^{n+1}=E^nE=EE^n$.

兹定义下列新操作。

(iii) E^*F.

定义

$$E^*F=F\vee EF\vee EEF\vee EEEF\vee\cdots=\sum_{n=0}^{\infty}E^nF,$$

或在形式上有

$$=(1-E)^{-1}F,$$

卽当 F 发生后，E 从 0 次，1 次，2 次，…… 地发生，换言之，在 F 发生后，E 以外的事件不会发生。

于是，以确定性事件作为单位，凡适用上述三个规則：$E\vee F$，EF，E^*F 多次結合得到的任意事件，称为**正規事件** (regular event)。由于有 E^*F 的結合法則，所得到的事件，一般地不是确定性的，因此，一般非确定性的事件，也許偶而有可能用这个方法构成有限形式的表現(**正規表現**)。事实上在某一范圍內，一般是可能的，如下节所示。

正規事件的例 为了帮助理解正規事件的意义，以下举若干例子。此处，X_1 表示“輸入 X_1 为 1”的事件，X_1' 表示“輸入 X_1 为 0”的事件。I 是恒等事件，卽恒为 1(不論輸入如何)的事件。以上长度均为 1.

(i) I^kI^i	$s=k+1$(“在开始动作后再恰好經过 k 次”).
(ii) I^k	$s\geqq k$(“在开始动作后，再經过 $k-1$ 次以上”).
(iii) $(I^3)^*I^i$	$s\equiv 1\pmod 3$.
(iv) $I^i\vee II^i\vee I^2I^i$	$s\leqq 3$.
(v) I^*X	“$X=1$ 至少发生一次”.

(vi)	$X^{*}X^{i}$	“經常 $X=1$”.
(vii)	$(X')^{*}(X(X')^{*}(X')^{i}\vee X^{i})$	“$X=1$ 仅发生1次”,或写成 U.
(viii)	$((X')^{*}X(X')^{*}X)^{*}U$	“$X=1$ 奇数次发生”.
(ix)	$X_1'^{*}X_2$	“当最后 $X_2=1$ 以后,所有 $X_1=0$”.
(x)	$I^{*}XI^{2}I^{i}$	“当 $t=4$ 时, $X=1$”.
(xi)	$I^{*}XX'^{2}X'^{i}$	“当 $t=4$ 时,才开始 $X=1$”.

从上述例子可知，正規表現是正确而簡便地表現各种事件的形式，并能精密地区别事件的“現在”和“过去”,“現在結束”等情况。可是,如下文所述,也有不能用此形式表現的事件。

§36 有限自动机的状态的正規表現

从上述例中知道,可簡单地用語言表示的大多数事件,能得到正規表現,因而是正規事件。后文将叙述,实际上存在着尽管可用語言簡单地表示,而不是正規事件的情况。正規事件的重要性质是,它可以用有限个元件构成,因而也是事件由有限的內部状态的自动机輸出所表現的充分与必要条件。下面将証明主張这种看法的两个定理。

定理 36.1 自动机具有有限个內部状态,从确定的状态 S_1 出发,在时刻 s 进入状态 S_i, 这样的事件是正規事件。

这个定理的意义在于,自动机从 S_1 出发,相継迁移至 $t=s$ 时,状态 S_i 产生,則此輸入系列的全体的集合是正規表現的。証明时,代替輸入系列,可直接处理从某一状态向下一状态的迁移,即从 S_1 至 S_i 的状态迁移鏈。并从此种迁移鏈的集合的表現出发导入正規表現。

将状态 S_i 向 S_j 直接**迁移的集合**写作 T_{ji}. 若不是这样的事則是空集；若有若干輸入信号产生 $S_i \to S_j$ 迁移，則它們都包含在 T_{ji} 中。

下面，对于**迁移鏈的集合**，将导入三个**結合規則**。

A_{ji}, B_{ji} 都是从 S_i 至 S_j 的迁移鏈的某个集合，則 $A_{ji} \vee B_{ji}$ 具有和集的意义。

A_{ji} 是 $S_i \to S_j$ 的迁移鏈的集合，B_{kj} 是 $S_j \to S_k$ 的迁移鏈的集合，而 $B_{kj}A_{ji}$ 是将各个集合所属的迁移鏈結合得到的 $S_i \to S_k$ 的迁移鏈的全体构成的集合。

更进一步，B_{jj} 也认为是 $S_j \to S_j$ 的迁移鏈的一个集合，A_{ji} 认为是 $S_i \to S_j$ 的迁移鏈的集合，而 $B_{jj}^* A_{ji}$ 是按照属于 A_{ji} 的迁移从 S_i 到达 S_j 后，再按照属于 B_{jj} 的迁移鏈(同一种或别一种均可)多次回轉(0 次也包含在内)而回到 S_j 的迁移鏈的全体集合。

对 T_{ji} 多次使用以上三个規則，得到的迁移鏈的集合的表現，称为正規表現。于是下面的补充定理成立。

补充定理 設有有限状态 $S_1, \cdots, S_M$，当給出 $S_i \to S_j$ 直接迁移的全体集合 T_{ji} 时，則任意二状态的从 S_i 向 S_j 的迁移鏈的集合的正規表現存在。

用数学归納法証明。首先 $M=1$，即只有一个状态时，記作 S_1, T_{11} 是 $S_1 \to S_1$ 的直接轉移，因此，表示 $S_1 \to S_1$ 的所有迁移是 T_{11} 的反复，这集合是

$$\Gamma_{11}^{(1)} = T_{11}^* T_{11}.$$

其次，状态数为 $M-1$ 时，这个定理仍然正确，仅将其中自某状态 S_i 向 S_j 的只經过 $M-1$ 个状态的迁移鏈的集合的全体写作 $\Gamma_{ji}^{(M-1)}$.

于是，在这里添加一个状态 S_M 时，产生一新的 $S_i \to S_j$ 的迁移鏈，显然至少包含一次 S_M. 一般經过 k 次 S_M 的鏈是如下的形式：

$$S_i \to \underbrace{S_M \to S_M \to \cdots \to S_M}_{k} \to S_j,$$

迁移包含两部分，一部分为将 S_M 除去，作为 $M-1$ 个状态中通达的鏈，另一部分是 $S_M \to S_M$ 的直接迁移。此处，$S_M \to S_M$ 的部分写成

$$A_{MM} = \sum_{k,h}^{M-1} T_{Mk} \Gamma_{kh}^{(M-1)} T_{hM} \vee \sum_{k}^{M-1} T_{Mk} T_{kM} \vee T_{MM},$$

$\sum$ 的意思是指参数的所有数值的 $\vee$ 总和。

同样，可写出

$$S_i \to S_M \quad 是 \quad A_{Mi} = \sum_k T_{Mk} \Gamma_{ki}^{(M-1)} \vee T_{Mi},$$

$$S_M \to S_j \quad 是 \quad A_{jM} = \sum_h \Gamma_{jh}^{(M-1)} T_{hM} \vee T_{jM},$$

結果

$$\Gamma_{ji}^{(M)} = \Gamma_{ji}^{(M-1)} \vee A_{jM} A_{MM}^{*} A_{Mi},$$

$\Gamma_{ji}^{(M)}$ 仅是 $\Gamma_{ji}^{(M-1)}$ 和直接迁移应用上述三个規則的表現。

以上是 i, j 都不是 M 的情况，設 $j=M$，

$$S_i \to S_M \to S_M \to \cdots \to S_M,$$

則

$$\Gamma_{Mi}^{(M)} = A_{MM}^{*} A_{Mi}.$$

設 $i=M$ 时

$$S_M \to S_M \to \cdots \to S_M \to S_j,$$

則

$$\Gamma_{jM}^{(M)} = A_{jM} A_{MM}^{*} A_{MM} \vee A_{jM}.$$

又 $i=j=M$ 时，

$$\Gamma_{MM}^{(M)} = A_{MM}^{*} A_{MM}.$$

以上都是三个規則的应用，因此証明了补充定理。

在定理 36.1 的証明中，可用相当于 $S_i \to S_j$ 迁移的輸入信号代替从补充定理得到的正規表現 T_{ji}，并用表示长度为 1 的輸入系列的集合的符号代入。并且知道所得的 $\Gamma_{ji}^{(M)}$ 正規表現中，其中时

間上最初的因子(即乘积右边的因子)定为初期事件，在肩上标有 i. 如此获得的式子正是表示由状态 S_1 到 S_i 所必需的輸入系列的正規表現。

以上是定理 36.1 的証明，这样获得的表現，一般非常复杂。在所有长度为 1 的輸入系列的符号組成中，可能含有很多空集。若除去含有空集的项，稍可簡单一些，再利用各种恒等式进行簡化。但是从定理 36.1 得到的表現，能否一定化成直接表示机器本来机能的形式是并不明确的。重要的是，正規表現能将有限个元件組成的自动机的动作充分地表現出来。

§37 正規事件的物理实现

下面打算說明这个逆定理：正規事件总可以用有限个元件(例如变參元件)組成的自动机的輸出，也就是使某一个元件的状态为 1 的条件来实现。此定理的証明也是对給定所要求的机能的自动机的**綜合**問題的一个答案。

这个証明的根本思路是极簡单的。即实現 E, F 的綫路已經組成时，就可表明由此实現 $E\vee F$, EF, E^*F 綫路的組成規則。然而，这里也有困难的地方，即执行物理中的 $\vee$, & 等邏輯操作时，必然有一定的时間延迟，在有环回路的場合，邏輯操作的时間很长，因此造成不利的因素，要把定理严密地証明将非常困难。但是，放弃一般性，下面定理的証明是能滿足的。

定理 37.1 当給定一个事件的正規表現的形式后，若适当地給定正整数 p, δ，并使此事件的延迟为 δ/p，那末表示 $s+\delta/p$ 时刻的輸出状态的变參元件网絡是存在的。

証明 这个問題中困难的是邏輯操作必然带有延迟的处理。首先，将 $\vee$, & 等邏輯操作所带有的延迟忽略掉，然后再考虑其后果。

令 $\mathscr{N}(E)$, $\mathscr{N}(F)$ 表示以 E, F 作为輸出的网絡，則問題变为研究如何构成 $E\vee F$, EF, E^*F 网絡。如果取得成功，則由于已知构成长度为 l 的确定性事件的綫路的方法（§35），即可以获得这种綫路。

（i）$E\vee F$.

只要将 $\mathscr{N}(E)$ 及 $\mathscr{N}(F)$ 的邏輯和取出即可(图 37.1).

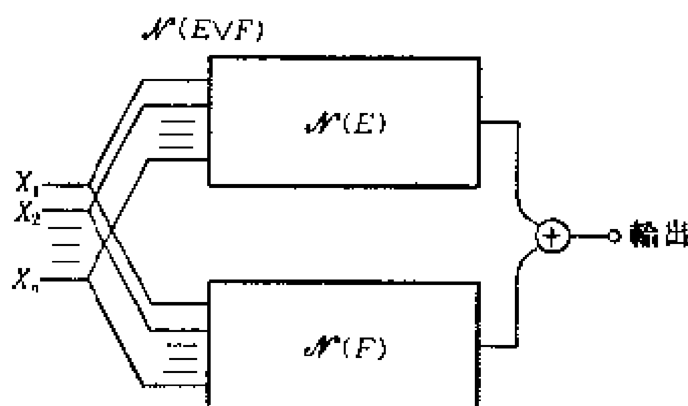


图 37.1 $E\vee F$ 的表示

（ii）EF.

E 事件是正规表現，其“**原始单位**”的定义可由下述归納的定义給出。

(a) E 本身是一个单位，則其原始单位即其自身。

(b) $E=E_1\vee E_2$, 則 E 的原始单位是 E_1 及 E_2 两方面原始单位之和。

(c) $E=E_1E_2$ 或 $E=E_1^*E_2$, 則 E 的原始单位是 E_2 的原始单位。

所以，若 E 的原始单位是 U_1, U_2, $\cdots$, U_n, 則认为 E 是 U_i, E_iU_i, $E_iF_i^*U_i$, $E_iF_i^*G_i^*U_i$, $\cdots$ 等将右边 U_i 的項以 $\vee$ 联結表示。

而表示 EF 的綫路的构成要借助于 $\mathscr{N}'(E)$. 这个 $\mathscr{N}'(E)$ 是由图 37.2 的綫路实現的。就是将 E 的“原始单位”U_1, U_2, $\cdots$, U_n 通过 $\mathscr{N}(U_1)$, $\mathscr{N}(U_2)$, $\cdots$, $\mathscr{N}(U_n)$这些部分綫路的輸出处和

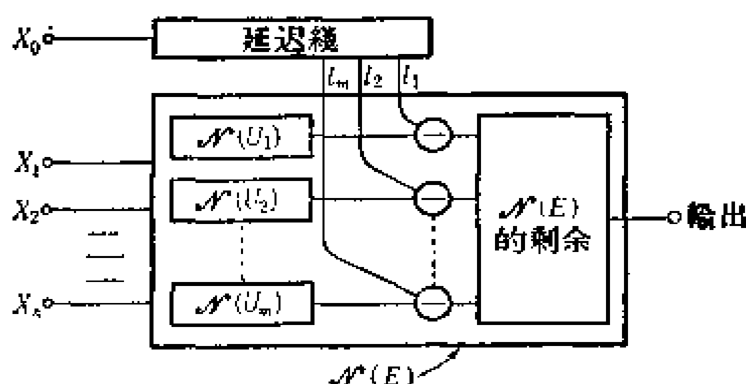


图 37.2 $\mathcal{N}'(E)$的构成

X_0 的各級等长度延迟相交，而得到 $\mathcal{N}(E)$ 的剩余部分，即为 $\mathcal{N}'(E)$. 将 $\mathcal{N}(F)$的輸出作为 $\mathcal{N}'(E)$的 X_0 的輸入，則 EF 的綫路組成如图 37.3 (a).

(iii) E^*F.

此处与(ii)同样要借助于 $\mathcal{N}'(E)$. 图 37.3(b) 中将 $\mathcal{N}(F)$ 与 $\mathcal{N}'(E)$ 的輸出由邏輯和組成，并将此邏輯和作为 $\mathcal{N}'(E)$ 的

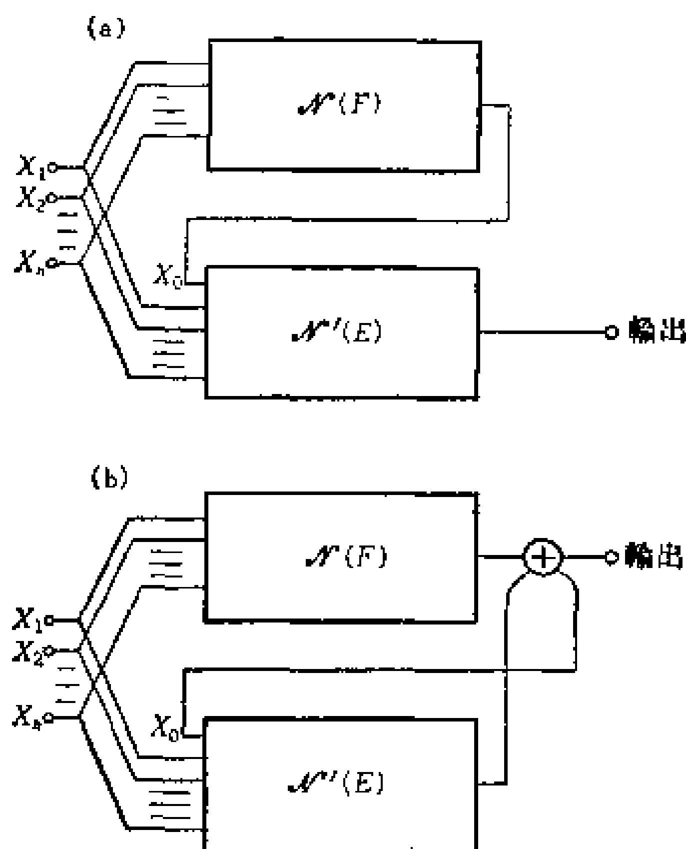


图 37.3 (a) EF, (b) E^*F 的表示

X_0 輸入，經过 l 长度的延迟，在邏輯和的輸出处得到 E^*F，其中将分别出現 F, FE, FEE, …的邏輯和。* 操作是对应非确定性的事件，图中必須具有反饋的构造。

延迟的处理 实际的元件，例如变参元件等，每通过一次必定产生若干延迟。象图 37.2 中，邏輯和元件后面連有長的延迟綫，邏輯操作的延迟，作为延迟綫延迟的一部分，因此并不产生問題。而图 37.1 中，取多数事件的邏輯和时，若后續事件的(原始事件的)长度較短，邏輯操作的延迟較长，則即使把延迟綫全部算入也嫌不够。

在后續綫路中全体增加延迟也是方法之一。然而，假如在輸入变数 X_1, X_2, … 处插入长度为 δ 的延迟，則輸出同样增加延迟 δ, X_0 的輸入亦延迟 δ，正好吻合。EF 的綫路中，允許 F 的輸出延迟較多，若允許輸出有很大的延迟，則由 E, F 組成 EF 时，所产生的困难就消除了。

同样，E^*F 的綫路中，若允許輸出有較大的延迟，則困难亦可解决。然而，将 $\mathcal{N}(E)$ 的輸出直接返回 $\mathcal{N}(E)$ 的輸入，形成环状的信号通路时，情况就复杂了。必須将环狀的信号在各种邏輯操作后給予一个周期的延迟，再返回輸入，这就增加了以"周期" p 为长度的延迟，以补救时間的差异。

为了避免环狀綫路增加过大的延迟，这里可适当地加大周期 p，以解决一些問題。p 增大，时間单位亦增大，环作用一周的时間必須縮短，以便返回时不必延至下一周期。

彻底的解决办法是实行所謂单位的重新組合 (reorganization)，环由很长的单位构成，而在延迟綫部分中扣去邏輯操作所需的时間。但这样就需要很多麻煩的手續，此处从略。这个方法的理由可参閱 Kleene 的論文。[Kleene 使用的神經原元件(neuron element)的多数輸入的邏輯和可以是一級的，这一点和变参元件有差异。]

綫路的綜合問題 本节的結論是針对实际要求給出的，它是

属于应用綫路設計方法的性质，亦即，在实际的問題中，有不少場合对輸入是有很多限制的。有很多輸入并不引起綫路的反应，这样，可以要求輸入信号分为三个部分：

部分集合 A 是属于形成输出为 1 的那些輸入；

部分集合 B 是属于形成輸出为 0 的那些輸入；

部分集合 C 是属于不論輸出是什么都可以的輸入。

将此方法应用于綫路的設計，灵活性很大，因此很難建立一般适用的方法。对于这类問題不采用一般性的处理方法。

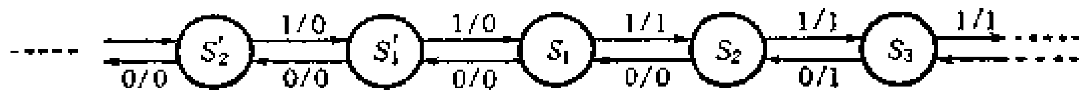


图 37.4 具有无限多内部状态的自动机

非正規事件的例子 具有 0, 1 两种輸入信号的自动机中，考虑下列事件："从开始到现在，1 的出现比 0 的多"。要判断，必須比較 1 与 0 的差数，若差数是正的，并且很大，则欲真正表現上述事件，必须这个机器有很大的数，仅由有限个元件构成的有限个内部状态的机器是不可能表示它的。上述事件根据定理 37.1 是非正規的。这种事件在下一节 Turing 机器中将表示出来。

这里，用語言表达起来很簡单的事件中也存在着非正規事件，这是很有趣的。

第3章　Turing 計算机

§38　Turing 机器与計算

至今所考虑的都是以有限个元件构成的、具有有限个內部状态的自动机。他們只有有限的信息存貯容量，因此机器的能力受到限制。要能超过此种限制，以便执行我們日常所想到的所有的計算及邏輯判断等工作，必須使机器具有**无限的存貯(記忆)容量**。当然，无限容量的存貯装置并非一开始就存滿信息的，例如，用机器以十进制小数計算 π，无非是在一定的时間內只用一定的存貯容量，計算出一定的位数。假如計算时間延長，多用存貯量，則可算得更多的位数。使用两端无限长的带子(紙带或磁带)作为无限的存貯装置的机器，称为 Turing 計算机或 Turing 机器。

Turing 計算机有三个部分，即由有限自动机构成的**計算机本身**，将带子写入、讀出并使其左右移动的**輸入輸出装置**和无限长的**带子**。自动机在一个单位时間內所执行的是，从带子上讀出符号，作为机器輸入，在这个信息的基础上給出輸出信号，并向带子上写入这个新的符号(从前的符号消去)，再将带子向左或向右移动一格。

所謂 Turing 計算机是这种机器的总称，实际种类很多。Turing 机的特征之一是它的計算机本身是有限自动机性质的，能用迁移图或迁移表完全描述其性能。迁移表如表 38.1 的形式，左面的二行标示“初态”及輸入即“讀符号”，接下去是輸出即“写符号”和讀写头“移动方向”，最后是“新的状态”。Turing 称如此一行的內容为指令。

表 38.1

初　态	讀出符号	写入符号	移动方向	新的状态
S_1	0	0	R	S_1
	1	2	L	S_3
	2	2	L	S_3
S_2	0	0	L	S_2
	1	1	L	S_4
	2	0	L	S_5
⋮	—	—	—	—

Turing 考察这种机器的目的是为了研究机器进行計算工作的实质，他不先导入各种計算的内容，而从純理論的角度去考察，并导入**計算表** (computable number) 的概念，以处理在数学基础理論中所提出的問題。各种各样的計算，从理論的角度去考察时，有在头脑中进行各种“心算”的同时，还須借助于数字及符号在紙上书写計算。心算是属于有限自动机的活动，紙是作为外部存貯(記忆)，以补助头脑存貯量的不足。当然在各种計算进行时，紙是作为二度空間被利用的，但这不是本质問題，在一度空間的带子上記录数字，虽多少有些不便，但仍可照样計算。

总之，在計算中所必需的設备是：相当于紙張的带子，可在带子一个区域(格子)内讀出或写入的装置(讀写头)，将这个讀写装置移动到带上任意位置的装置，以及控制以上各装置并且能执行基本演算規則的机器本体。实际上，不一定是讀写头在带上移动，带子相对于讀写头向左或右作一格移动也同样可以的。上面是叙述 Turing 計算机的构思。

一度空間的带子用于书写記忆时，下述二个内容是基本的。

(1) 带子的某一部分有一串符号(字)，将他們**移动**到别处；

(2) 带子的某一部分有一个字，将带子某部分中同样的字**探**

索出来。

如果上述两点能满足，則带子的作用与各种紙的作用一样，能满足存貯复杂的計算中所給予的任务。实际上，假如計算机本身有适当的机能，上述要求也是办得到的。例如在(1)的情况下，将字一位一位地移动，再加上表示已經移到哪一位的标記，每移动一位使标記也变动一步卽行。又例如在(2)的情况下，可以一位一位地比較，先把最初一位相同的字全部加上标記，然后在有标記的字中探查第二位相同的字，再消去以前的标記，加上表示两位相同的字的新标記。这样一位一位地比較，直到各位都相同的字全部找出为止。

§39 Turing 計算机的标准化

在 Turing 計算机中，根据任务的不同，可以从簡单到复杂有多种型式。Turing 机的复杂性可以用一格內所用的符号数目(种类)N，和計算机本身的內部状态数 M 作为大体上的指标。前述的迁移表的行数，卽"指令"的数就是 N 和 M 的乘积，这个乘积 MN 也許可以作为复杂性的指标。

一般情况下，同样的机能，卽能进行同样的数的計算的机器，其构成的方法并不是一定的。特别在使用較大的符号数目 N 时可用較少的內部状态数 M，而 M 大时，N 可較小。Shannon 証明了，对于一个 M, N 一定的 Turing 机，可以构成一个只用二个符号的 Turing 机，在全部机能上跟它等价，并且也可以构成一个只有二个內部状态的 Turing 机跟它等价❶。

(I) **仅用二个符号**(空白(0)或1)**的机器**($N=2$)　在带子的一格中只写0或1，二者必居其一，連續 n 格組成一組，若将其作

❶ Shannon 的論文发表在 Proc. of London Math. Soc., No.(2)**42**, pp. 230~265. ——譯者注

为二进制数，則和 $0\sim2^n-1$ 个数的二进制符号是一致的。因为机器不能把 n 格的內容同时一起讀入，所以从一端例如从左端开始串联讀入，机器本身应能够寄存这 n 格的內容（作为内部状态）。在写出时，同样应将机器內部状态（現在是从右端开始）一位一位地写上带子。

現在具体說明，一个具有 M 个內部状态、$N(\leq 2^n)$ 个符号的 Turing 机 $\mathfrak{M}$，是和仅用二个符号的 Turing 机 $\mathfrak{M}'$ 在机能上等价的。

(1) 要証明这一結論，只需找到 $\mathfrak{M}'$ 的状态 $T_1, T_2, \cdots, T_M$ 数与 $\mathfrak{M}$ 的状态 $S_1, S_2, \cdots, S_M$ 是一一对应的即可。下面就叙述 $\mathfrak{M}'$ 通过什么办法，来利用 0, 1 二个符号获得 M 个状态。

設 $\mathfrak{M}'$ 处于从带子上 n 个格子的最左格子开始讀出时的状态。假如讀到的符号是“0”，則讀头向右移动一格❶，并且机器的状态由 $T_i \to T_{i0}$. 假如讀到“1”，讀头仍向右移动一格。机器的状态变为 T_{i1}.

(2) 若下一个讀到“0”，則状态由 $T_{i0} \to T_{i00}$，或由 $T_{i1} \to T_{i10}$，若下一个讀得“1”，則状态由 $T_{i0} \to T_{i01}$，或由 $T_{i1} \to T_{i11}$，讀头再向右移动一格。

同样，(3)，(4)，…按上述步驟进行。

$(n-1)$ 在第 n 格右端前一格所讀到的状态是 $T_{i,x_1,x_2,\cdots,x_{n-1}}$ 的形式，讀头再向右一格即达到端点。这一阶段的状态数最大为 $M\times2^{n-1}$ 个。

(n) 这个步驟是 $\mathfrak{M}'$ 的特征步驟。在讀到右端符号 x_n 时，机器根据其內部状态和当时讀得的符号，即得到与 $\mathfrak{M}$ 相当的“讀符号”。以 $\mathfrak{M}$ 的迁移表为准，$\mathfrak{M}'$ 运轉一周，向其新的状态 $L_{k,y_1,y_2,\cdots,y_{n-1}}$ 或 $R_{k,y_1,y_2,\cdots,y_{n-1}}$ 移动。L(左) 还是 R(右) 由 $\mathfrak{M}$ 的迁移表所給出的讀

❶ 亦可使带子向左移动。——譯者注

头运动方向决定。这里,$(y)=y_1, y_2, \cdots, y_n$ 是 $\mathfrak{M}$ 迁移表中的"写入符号", k 是"新的状态", $\mathfrak{M}$ 处于状态 S_k。同时机器在这里,即最右端的格子上印刷符号 y_n,讀写头向左移动一格。

$(n+1)$　現在写下符号 y_{n-1},讀头向左移动一格,状态变为 $L_{k, y_1, y_2, \cdots, y_{n-2}}$ 或 $R_{k, y_1, y_2, \cdots, y_{n-2}}$。

下面同样向左运动,逐次填上新的符号(0 或 1)[1]。

$(2n-1)$　讀写头重新回到一个字的最左端的格子时,填入符号 y_1,状态是 L_k 或 R_k,此时写入工作完毕。讀写头向左或右移动 n 格,到达与 $\mathfrak{M}$ 机器相邻格子相当的部分。因此,必須要有 $(n-1)M$ 个状态 U_{ks} 或 V_{ks} $(k=1, 2, \cdots, M;\ s=1, 2, \cdots, n-1)$ 来表示。

其向左移动的步驟是

$$L_k \to U_{k,1} \to U_{k,2} \to \cdots \to U_{k,n-1} \to T'_k,$$

或向右移动的步驟是

$$R_k \to V_{k,1} \to V_{k,2} \to \cdots \to V_{k,n-1} \to T'_k,$$

結果达到 T'_k 状态。再根据 $\mathfrak{M}$ 的下一步驟进行相应的动作。

以上 $(3n-2)$ 步动作相当于 $\mathfrak{M}$ 机器的一个动作,$\mathfrak{M}'$ 的状态总数中 T', L, R 状态的总数是 $3M(2^n-1)$,U, V 的总数是 $2M(n-1)$。$N=2^n$ 时,$\mathfrak{M}'$ 的状态总数約为 $3MN$ 个。这是符号数减少为 2 的代价。即使 N 不是为 2 的方冪,如以适当的二进制碼表示,大致 $3MN$ 个状态也可办到。

(II) **仅有二个內部状态的 Turing 机** $(M=2)$　現在以少量的內部状态,使机器內部的寄存設备减至最少,而以带子上增加記录內容来得到与任意的 M, N 机器等价的功能。

新的机器 $\mathfrak{M}'$ 只有二个內部状态,設为 α, β. 它是带子上一个格子向下一格子运送信息的唯一手段。

---

[1] 旧的擦去。——譯者注

最初是相当于 $\mathfrak{M}$ 的一个动作的 $\mathfrak{M}'$ 的一个周期的动作，即 $\mathfrak{M}'$ 根据特定的操作表工作，写入新的符号，向右或向左移动。现在在这一个格子中写入的符号表示 $\mathfrak{M}$ 机器所应当写入的符号 A_i，$\mathfrak{M}$ 机器应当取的新状态 S_j，还有移动方向是右还是左 (R, L)，以及表示本格子是相邻格子发送信息(发送格子)的符号 T. 所以这符号的形式是 (i, j, L, T) 或是 (i, j, R, T) 的組合。

机器的下一任务是向右或向左移动，并在下一格子中記上它应向哪一边格子接收新信息的記号。机器按照迁移表，以状态 α, β 的区别来存貯这样的信息，即机器走到这下一格子后，見到符号 i'，根据 $\mathfrak{M}'$ 的迁移表，在这格子上写上表示"接收格子"的記号 R 和返回的方向左 (L) 或右 (R)，再添上一个数字 1，成为 $(i', 1, L, R)$ 或 $(i', 1, R, R)$.

再下一个任务是将发送格子的数字 j 迁入接收格子。这就是在此二格子間往复摆动，每摆动一次，使发送格子的数字 j 减 1，而接收格子的数字 j 加 1. 利用机器中所存貯的一位(bit)信息来表示操作的結束，即在发送格子的 j 变成 0 时，通知接收格子上述事实。

将发送格子的 j 迁移完毕后，因为以后这格将暂不应用，所以此格内不用 $\left(i, 0, \frac{L}{R}, T\right)$ 表示，仅写入 (i) 即可。这时在 α 状态下，轉向接收側操作，执行对应于 $\mathfrak{M}$ 的迁移表中的迁移并写入对应于 $\mathfrak{M}$ 的新状态的符号。于是对应于 $\mathfrak{M}$ 机器的一步过程执行完毕。

表 39.1 是 $\mathfrak{M}'$ 的迁移表，而且对应于 $\mathfrak{M}$ 的迁移表中的

$$S_j; A_i \to A_k, \frac{R}{L}; S_l$$

是 $\mathfrak{M}'$ 的特征迁移

$$\alpha; (i, j, x, R) \to \left(k, l, \frac{R}{L}, T\right), \frac{R}{L}; \frac{\beta}{\alpha}.$$

表　39.1

初　态	讀出符号	写入符号	移动方向	新的状态	
α	(i)	$(i, 1, R, R)$	R	α	$(i=1, 2, \cdots, N)$
β	(i)	$(i, 1, L, R)$	L	α	$(i=1, 2, \cdots, N)$
β	(i, j, x, R)	$(i, (j+1), x, R)$	x	α	$\begin{pmatrix} i=1, 2, \cdots, N \\ j=1, 2, \cdots, M-1 \\ x=R, L \end{pmatrix}$
α 或 β	(i, j, x, T)	$(i, (j-1), x, T)$	x	β	$\begin{pmatrix} i=1, 2, \cdots, N \\ j=2, 3, \cdots, M-1 \\ x=R, L \end{pmatrix}$
α 或 β	$(i, 1, x, T)$	(i)	x	α	$\begin{pmatrix} i=1, 2, \cdots, N \\ x=R, L \end{pmatrix}$

上述方法中，带子写出的符号数約为 $4MN$ 个。

Shannon 还曾論述了只有一个内部状态 $(M=1)$ 是不能成为有意义的計算机的，此处省略❶。

§ 40　通用 Turing 机

Turing 机根据机器本体的构造及带子上写入的一串初始信息，能作各种計算(但不一定是所有的計算)。所謂**通用 Turing 机**就是能进行"一切計算"的 Turing 机。令人难以置信的是，例如一个只有 1000 个元件的机器，和具有 2000 个元件的机器有相同的机能。但实质上也不奇怪，所謂通用机器是一种模拟性的机器，这种机器能将某一机器 $\mathfrak{M}$ 的操作規則适当地"符号化"，并記在带子上，使通用机器按照 $\mathfrak{M}$ 的操作一步一步地执行。那末将各种計算机的指令表編成計算的程序，通用机就能执行一切計算机的工作了。但是执行同一任务的速度，通用机可能不及 $\mathfrak{M}$ 快。

❶ Shannon 曾証明，只有一个内部状态的 Turing 机不可能是通用的，參閱前注文献。——譯者注

前节曾証明，存在一种只有二个符号的 Turing 机 $\mathfrak{M}'$，它的机能与任意的 Turing 机 $\mathfrak{M}$ 是等价的。下面給出一个 $N=2$, M 是任意的通用 Turing 机的例子，以便帮助理解 Turing 机的动作。此通用机器本身的 $N=6$, $M=12$.

通用 Turing 机举例 根据上述，$\mathfrak{M}$ 是以 0(空白)，1 二种符号构成而内部状态为任意多的 Turing 机，而 $\mathfrak{M}'$ 是以 0, 1, ;, *, §, ¶ 六种符号(当然符号的形式不管怎样都可以)及 A, B, …, L 12 个内部状态組成的 Turing 机，$\mathfrak{M}'$ 模仿 $\mathfrak{M}$ 的工作。$\mathfrak{M}'$ 的迁移表如表 40.3，迁移图如图 40.1.

首先說明如何将 $\mathfrak{M}$ 的"說明书"写在带子上。設 $\mathfrak{M}$ 的迁移表如表 40.1:

表 40.1

初态	讀出符号	写入符号	移动方向	新的状态
S_1	0	0	R	S_2
	1	1	R	S_5
S_2	0	1	L	S_3
	1	1	R	S_2
⋮	⋮	⋮	⋮	⋮

则在 $\mathfrak{M}'$ 的带子上应包含着这个表的内容，标題的部分(即左 2 列)，仅将与内容相应的部分，按上表順序，自右至左每一行以；分开写入。"新的状态"是 S_r，以 $2(r-1)$ 个 1 表示。再者，輸出即相应于"写入符号"和"移动方向"的信息，以表 40.2 上規定的符号記入上述 1 列的右边。

表 40.2

写入符号	移动方向	𝔐′带的符号
0	L	(无)
1	L	0
0	R	00
1	R	01

上述例子表示如下：

$$\P\cdots;\underbrace{1101;11110}_{S_2};\underbrace{1111111101;1100}_{S_1}\P$$

以¶加在說明书的两端，作为記号。这里用到0, 1, ;, ¶四种符号。若在𝔐的带子上原有写入信息，则写在𝔐′带子的右边的¶的右边。然而𝔐的带子仅一边是无限长的，对于使用两边无限长的带子的Turing机，可經过适当的变换，把带子"折回"，即可成为仅有一边是无限长的(証明略)。

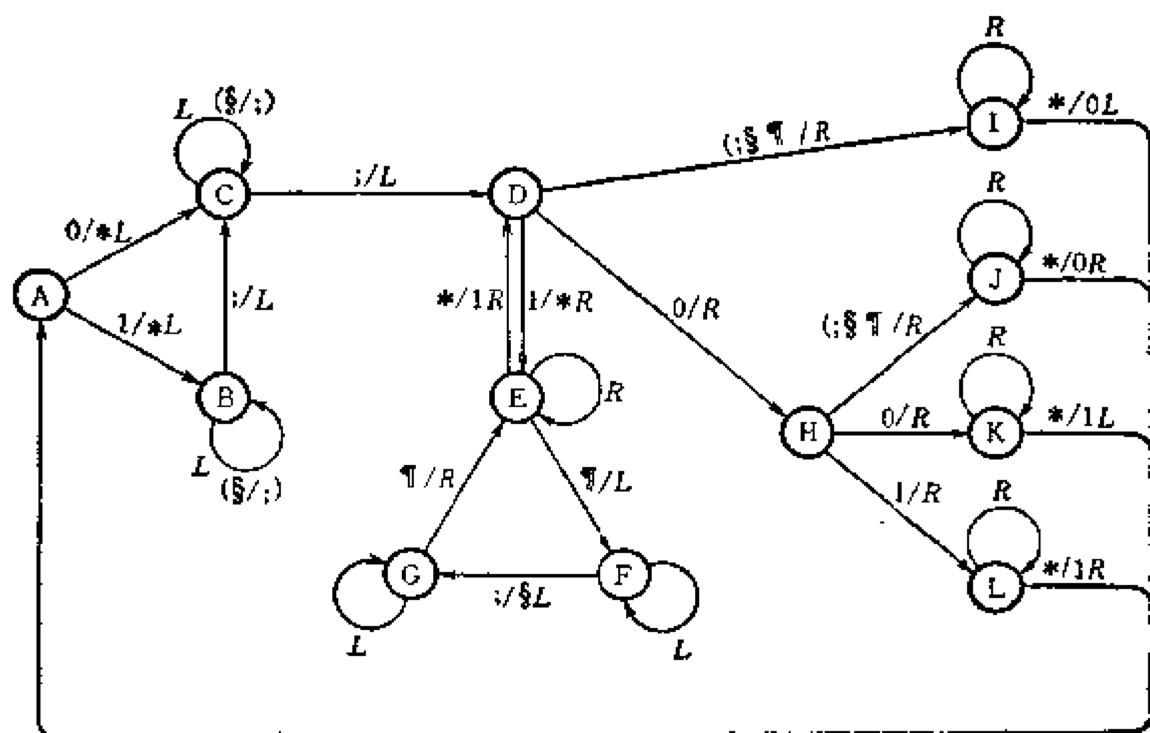


图 40.1 通用 Turing 机的迁移图

再者,約定在 $\mathfrak{M}$ 計算开始前的初态是 S_1。

在 $\mathfrak{M}'$ 带子中右边的 ¶ 的右边(称为 W 区域)写着与 $\mathfrak{M}$ 带子上相同的内容,这是为了进行計算。計算就是讀写头在 W 区域把 $\mathfrak{M}$ 带子的符号讀出,进行到"說明书"的部分(称为 I 区域),把必要的記載事項記存下来,再轉回 W 区域进行写入新的符号的一个行程。这样反复地进行計算。

$\mathfrak{M}'$ 的操作开始是这样的,例如状态是"A",讀写头在 W 区域上相当于 $\mathfrak{M}$ 带子最初讀到的地方。$\mathfrak{M}'$ 按照迁移表(表 40.3)动作。符号 *, § 是用于动作途中的标記。

表 40.3

A	0	*	L	C
	1	*	L	B
B	;		L	C
	§	;	L	B
	其他		L	B
C	;		R	D
	§	;	L	C
	其他		L	C
D	1	*	R	E
	0		R	H
	;, §, ¶		R	I
E	¶		L	F
	*	1	R	D
	其他		R	E
F	;	§	L	G
	其他		L	F
G	¶		R	E
	其他		L	G
H	;, §, ¶		R	J
	0		R	K
	1		R	L
I	*	0	L	A
	其他		R	I
J	*	0	R	A
	其他		R	J
K	*	1	L	A
	其他		R	K
L	*	1	R	A
	其他		R	L

12 个状态可大致分为三組,即

A, B, C　将 $\mathfrak{M}$ 的輸入取得的手續。

D, E, F, G　标記 $\mathfrak{M}$ 的"新的状态"的手續。

H, I, J, K, L　取得 $\mathfrak{M}$ 輸出的手續。

首先,第一部分是:

（i）**輸入手續** A～C

状态 A，将 $\mathfrak{M}$ 带子讀出，并在此位置写上标記 *，并根据讀出 1 或 0，迁入 B 或 C 状态，并把讀得的符号存貯。其次向左移动，在 § 的地方不改变并寻找最初的；．假如輸入为 0，它就是下一个指令的位置。B 状态的情况和 C 状态的不一样，要探查更左方的；，将第二号；查到，它就是輸入 1 时的下一个指令位置。如此，0，1 的区别能把讀写头引导到正确的指令位置。向右行是 D 状态，将最初的 1 变为 * 标記。

（ii）**标記新状态的手續** D～G

这一串动作是决定讀写头在二个 ¶ 之間作几次往返。在这里将

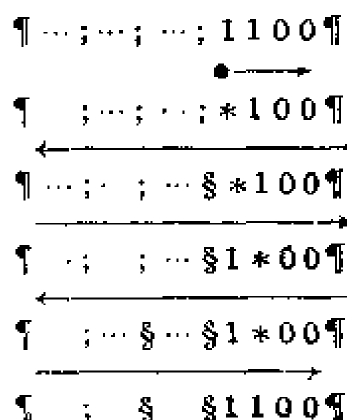
$$G \to E \to D \to E \to F \to G$$

的循环过程执行，在写上下一个“指令”的部分加以标記。

在 G 状态下达到左边的 ¶，变成 E 状态，向右探查 *，查到后变为 D 状态，把这里的 * 变为 1，并在右边将 1 变为标記 *．即把 * 向右移一格再轉为 E 状态，停到右端的 ¶ 并成为 F 状态。

現在是 F 状态，向左移动搜寻；标記。寻到后，将其变为 §，变成 G 状态并走到最左端。每一次往返在右端顺次将一个；变为 §.

将此模型的例子，表示如下：



H 状态→K 状态，向 W 区域移动，此时将 S_2 指令的标記写上。

在具有 $2(r-1)$ 个 1 的列上，經过 * 的变动一次，就有一个；变成 §，在 * 向最后一个 1 的归路上，右边有 $2(r-1)$ 个；变成 §，而"新的状态"则和从右看設有标記的最初二个；的右边指令有关。

(iii) **輸出手續**　H～L

在 E 状态时，发現最后的 1 的地方出現 *，則碰到 1 就轉入 D 状态，并在相邻处調查輸出符号，如果沒有 *（不外乎；，§，¶）就轉向 I 状态，假如碰到的是 0，則轉到 H 状态并向右边的字調查。右边无字則轉入 J，如果是 0 則轉入 K，如果是 1 則轉向 L 状态。如此将 I，J，K，L 四种輸出記存，再向右边的 W 区域去找到 * 标記。当发現后，将各种适当的符号写下，再向适当的方向移动一步，就轉入 A 状态，进入了下一周期。

以上簡述了通用 Turing 机如何根据一个 Turing 机的說明书而工作的，目的在于理解这种机器。Turing 所考虑的通用机器是使用任意个数的符号和状态而构成的，与此处所述机器的机能还有一些差别。通用机器本身使用几百个状态，符号也有十几个，是相当复杂的，上述机器是仅用 12 个状态的簡单机器，却能模拟任何复杂的机器，且具有通用机器的性质，不免使人有些感到意外。

第4章　作为自动机的电子計算机

§41　电子計算机的組成

在第一編中所說的程序存貯型計算机，是一个复杂的自动机。若将計算机的輸入，即由紙带上讀出数或符号看成是自动机的輸入信号，而計算机的輸出，即穿孔的数字和符号看成是自动机的輸出，則輸出完全由輸入的数和符号系列，即程序所决定。再者，輸入不是作为时間的函数由外部給定，而是伴随着程序进行，根据輸入指令才开始輸入，并同时准备带子上的新信号的讀入，这一点和 Turing 机是相似的。在 Turing 机中，机器将带子向左或向右移动；但普通的电子計算机中，紙带只能向一个方向移动，机器只能控制紙带讀不讀而已，有的机器能控制讀到何处。

Turing 机的輸出是写在同一条带子上的，在讀出的場所又进行写入，并且新写入内容后使过去的内容消失。而各种电子計算机的輸出带与輸入带是分开的，因为穿过了孔不可能把孔再消除。虽然有以上这些差异，但仍可认为它是和 Turing 机相类似的。

普通的計算机，对輸入輸出的时間因素沒有严格的規定。但**实时計算机**(real time computer)例外，它由外界的物理过程直接接受輸入，进行計算后，又直接用作控制外界的事物，它是控制系統的一个环节。而普通"計算机"的特征是不受时間的制約。因此在規定計算机的机能方面，不象自动机那样必須将它的性能完全規定下来，只要規定輸入和輸出的关系，具体說，是規定用什么指令，必須作多少位数的計算等就将机器的輪廓給定了，它的基本結构也大体定下了。

程序存貯式計算机由§2所指出的五个部分构成,它們之間的关系有如图2.1所示的数据和指令的通道。这五个部分各是一种具有明确机能的自动机,相互結合,构成一个有机的整体。下面考察各部分作为自动机的必要机能。

(I) **存貯装置** 这是用于存貯大量信息的地方,而且一旦存入一个信息,只要不把它消去,就希望它长久保持。这样就构成一种具有很多內部状态的非确定性自动机。

存貯装置中存貯信息的方式有二种:**靜态存貯**,即各个物理元件分别保持一位(bit)的信息;**动态存貯**是以担当信息的物理变化在形成环状的存貯装置中循环不息地傳播而保存信息的。

存貯的信息是整齐地排列着的,若干位信息的集合組成一个"字",根据外面的"地址选擇信号",将一个字"**讀出**",即在保存的原始信息的条件下向外面发送信号,而将一个字"**写入**"是从外面向某一地址輸入信号,要求保存下来,当然,原来的信息就消失了。这两种操作必須能够自由地进行。

一个存貯器具有n个存貯容量,每一个字的长度为l位,向这个装置的輸入信息应有

$A_1 \sim A_\nu$ 地址选擇信号 ν 位 ($n=2^\nu$)

W 写入指令

R 讀出指令

$X_1 \sim X_l$ 被写入的信息(有写入指令时才有效)

而作为輸出有

$Y_1 \sim Y_l$ 讀出信息(有讀出指令时才出現)。

变参元件是能滿足这些机能的。作为动态存貯,需$3L$个变参元件組成环状延迟綫,再用一部分門組成从外部写入的控制。当然在串联型的存貯器中,輸入$X_1 \sim X_l$是从一端串行輸入的,讀出选擇門綫路(如图24.1)执行地址的选擇(图41.1)。

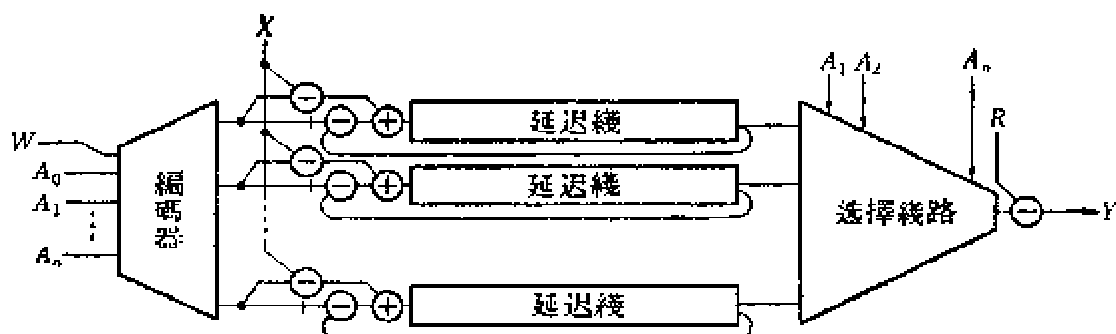


图 41.1 动态存貯装置

仅用变参元件构成大容量存貯器，从实用角度来看现实性不大。实用的动态存貯器使用水銀，固体(石英等)，金属絲(鎳綫)等利用声波的傳播組成**超声延迟綫**(图41.1)，其他也广泛地用**磁鼓**，**磁芯**組成静态存貯装置。

(II) **运算装置** 这是根据計算机的指令，执行加减乘除等各項运算的装置，也就是以被运算的数作为輸入，而以答数作为輸出的装置。若只从这个角度看，运算器是确定性的自动机，即純属邏輯綫路。但实际上乘除法是极为复杂的操作，不能一次获得結果，要交替使用加法和移位，因此通常是将中間結果折回，再行运算的有环綫路。

另一方面，被运算数或結果須"寄存"在运算装置内部，再者被运算数是从存貯装置中讀出的，通常要从存貯器中同时讀出二个字是不可能的，这就需要运算装置中具有暫时保存数字的設备。在运算装置中通常最少要有两个能寄存一个字的寄存器。

这就是将每次的計算結果保存，用作下一次被运算数的**累加器**，以及用作寄存乘法中乘数的**乘数寄存器**。乘法答数的位数有二倍长，正好放在累加器和乘数寄存器中，乘数的数字逐个使用以后，順次地排出，而将答数跟踪移入。同样，除法的被除数放在累加器中，除数放在乘数寄存器中是合理的。再者，当数字不能及时从存貯装置中讀出时(例如磁鼓，超声延迟綫等)，从存貯装置讀出

的数必須先暫时保留在运算装置中的**緩冲寄存器**中。在某些情况下，运算装置有更多的寄存器。

有些場合把运算器中控制乘除法串行操作的部分机构称为**乘除控制装置**。

(III) **控制装置** 这是計算机中最复杂的部分，統一控制計算机的**序列动作**(sequential operation)，并由若干功能部分組成。

計算机的操作是按程序中的一条条**指令**，順序地执行的。执行一个指令的一个周期可分为二个阶段。

[阶段 I] 从存貯装置把一条指令讀出。讀出哪一条指令？这个地址由具有計数綫路的**控制計数器**❶ (control counter) 給出。通过地址选擇器将指令讀出，并置入**指令寄存器** (instruction register)，同时将控制計数器的內容加 1.

[阶段 II] 从指令寄存器的地址部分，讀出地址并送到存貯器，按此地址讀出內容(如系磁鼓等存貯器，須先比較地址)。指令寄存器中寄存运算种类的部分联至一群由邏輯綫路构成的譯碼器(参看 §25)，各种运算各有一个对应的輸出信号端，这个信号将此指令执行时所需开关的各部分門加以控制，而各种指令要开关同一个門时，有必要用多端邏輯和組成編碼器(参看 §25)。这样，根据运算的种类，指令执行的內容如下：

加减乘除及讀出指令	数从存貯装置讀出，送到运算装置，向运算装置发出运算指示。
轉移指令	从指令寄存器的地址部分讀出，其內容送到控制計数器。在条件轉移时要由运算装置的信号来控制这个指令的实行。
写入指令	把累加器的內容向存貯装置写入。
移位指令	将累加器的內容向左或向右移动，指令寄存器

❶ 即順序寄存器。——譯者注

的地址部分表示移动的位数。

輸入指令　从輸入寄存器向运算装置傳送数碼，同时启动紙带的傳送机构。

輸入寄存器若沒有滿，指令处于等待状态；輸入寄存器滿了，指令执行。

輸出指令　从累加器向輸出寄存器傳送数碼，同时将穿孔及印刷机构启动。輸出寄存器空时，指令执行；否則，指令处于等待状态。

以上說明控制装置的工作。在这些操作中，讀出，写人，加减和轉移操作在极短的时間中就可作完，这些时間是一定的。将各个时間測定，就能得到**結束信号**，即发出下一次操作开始的指示信号。以上各部分的关系示于图 41.2 中。

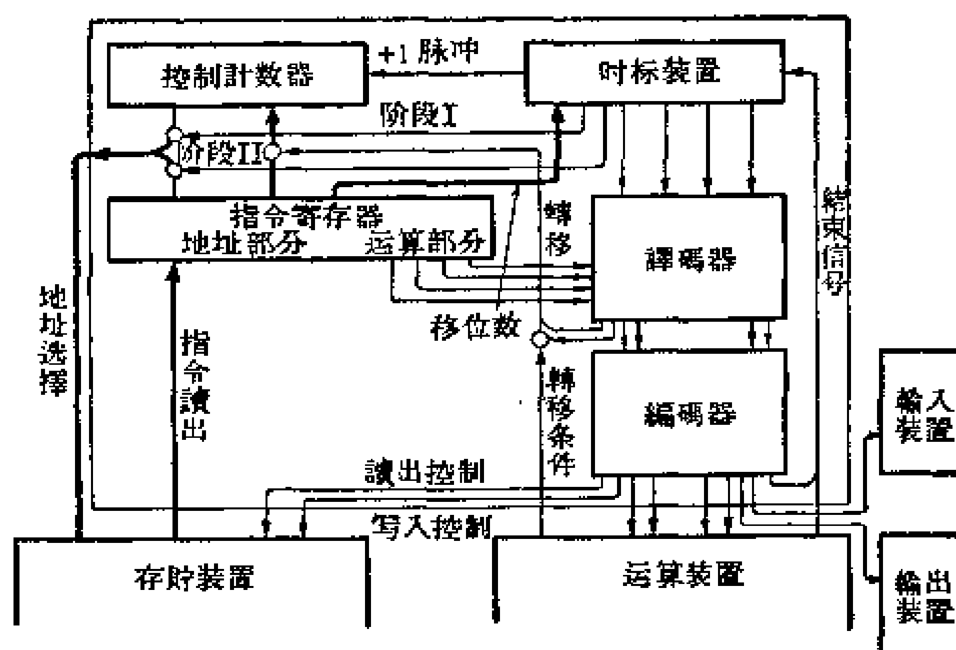


图 41.2　控制装置的組成

乘除操作所要的时間較长，而运算本身适当地进行串行操作是必要的。在这种情况下，有些机器是以一定的时間作完的，而有些机器的乘法，当乘数是 0 时不进行計算，根据乘数的有效位数，

机器所要的时間就不一样。这时，乘除控制装置在运算完毕时发出一个結束信号，控制器接受到这个信号后，再发出下一指令开始的信号。移位指令所需的时間也不一定。

輸入、輸出指令有**等待的机能**，这是一个特点。例如輸入指令所协同工作的是紙带讀取机，它是机械結构的，其动作的时間尺度与計算机本身完全不同，因此可以把它看成是电子計算机的外部設备。在輸入指令时，由輸入寄存器讀出时，同时撥动一个触发器，以表示輸入寄存器已“空”。同时作为自动机的“輸出”，給出一个启动紙带拖动机构的信号。

紙带启动，新的信号傳入輸入寄存器。当确认輸入寄存器已滿的时候，将上述触发器撥回，以指示輸入“准备完毕”。

輸入指令的結束信号，是在輸入寄存器向运算装置傳送完毕后发出的。輸入指令发出后，若輸入寄存器是空的，或在“准备完毕”以前，則操作不开始，从而結束信号也不发出。这时輸入命令应稍待片刻，因此在前面的善后工作处理好之前，若輸入指令接着再来就要让計算机等待。要使两种不同的时間尺度联系起来工作，这种“等待”的机能是必須的。

上述各动作的完成需要一定的时間，各个門的开关，必須按一定的时間关系进行，那末必須要有一个調整时間的**时标装置**。它根据“开始信号”起动，以后每隔一定的时間继續給出信号，产生种种的时标波形，直到終了。

串行型与并行型 計算机中的数和指令从一个地方送至别的地方，有各位分别从各个导綫傳送，即各位同时傳送的方式（**并行型** parallel system），以及只有一根导綫，各位的信号順次由这根导綫傳送的方式（**串行型** serial system）。同样，在运算装置中，数的运算也有各位同时計算的綫路（加减器），称为**并行运算装置**，以及只有一位运算装置，各位順次运算的**串行运算装置**。动态存貯

装置以串行方式比較合适,静态存貯装置以并行方式較合适,但也并不限于此。运算装置和数的傳送方式采取相同的型式較为普通,但也不尽是如此。运算装置以串行方法工作时,部件数量少,比較簡单,但缺点是速度較慢。通常并行方式用于速度高的高級机器,串行方式用于速度低的普及型机器。串行运算装置中,将数的低位先送(与普通的讀法相反)。加减法中从低位算起是为了进位的順序方便。

在串行計算机中(同步型綫路),除綫路的基本周期之外,还有一个单字巡回一次的时間周期(字周期 word cycle)。它是基本周期的 l 倍(字长 l),一位数字在一定的位置每隔 l 拍只出现一次。这里,对最高的数进入运算装置必須給于适当的指示,計算机必須具有以 l 周期連續循环的**"时計"**。这样,机器各部分的动作都和"时計"同步,数的傳送門的开关,只开放 l 周期,将一个字傳送。

§42 变参元件的运算綫路

計算机的各部分中,运算綫路是計算机所特有的,以下略述之。运算綫路按照

并行式或串行式

二进制或十进制

等区别而有各种方式。特别是用十进制数的情况,按照 0~9 的数字的編碼不同而具有各种不同的方式。最普通的編碼方式是二进化的十进制,即是以四位二进制数表示 0~9 数字的方法。其变形之一是增加 3 的"余 3 制"編碼(表 42.1)。

在十进制的情况下,若十进数字的四位二进制数是并行表示的,而十进数字本身是串行表示的,則称为**并串式**。其逆,二进制四位串行表示,而十进数各位同时表现,則称为**串并式**。完全串行或完全并行的亦可考虑,一般并串式用得較多。

表 42.1 十进制編碼

	純 二 进 制	余 3 制
0	0 0 0 0	0 0 1 1
1	0 0 0 1	0 1 0 0
2	0 0 1 0	0 1 0 1
3	0 0 1 1	0 1 1 0
4	0 1 0 0	0 1 1 1
5	0 1 0 1	1 0 0 0
6	0 1 1 0	1 0 0 1
7	0 1 1 1	1 0 1 0
8	1 0 0 0	1 0 1 1
9	1 0 0 1	1 1 0 0

二进制的优点是能以較少的数字表示較多的信息，更重要的是加減乘除的运算規則簡单。下面加以說明。

(i) **二进制加减法綫路**(并行式) 基本的二进制加法綫路如 § 23 的 III_{10}. 但在二进制中，二个数 $X_{n-1}X_{n-2}\cdots X_2X_1X_0$ 和 $Y_{n-1}Y_{n-2}\cdots Y_2Y_1Y_0$，其和 $Z_nZ_{n-1}Z_{n-2}\cdots Z_2Z_1Z_0$ 是

$$Z_0=X_0\oplus Y_0, \qquad W_0=X_0Y_0,$$

$$Z_1=X_1\oplus Y_1\oplus W_0, \quad W_1=(X_1,\ Y_1,\ W_0),$$

$$\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots,$$

$$Z_i=X_i\oplus Y_i\oplus W_{i-1}, \quad W_i=(X_i,\ Y_i,\ W_{i-1}),$$

$$\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots,$$

$$Z_n=W_{i-1},$$

基本上是 § 23 中的 III_7 和 III_{10} 的三变数函数，可是 III_7 函数可以在 III_{10} 中的一級上同时得到。图 42.1 构成这种加法器。

負数在运算綫路中，通常是以**补数**表示的。X 的二进制补数有二种：

"2 的补数" $2^{l+1}-X$ (补碼)，

"1 的补数" $2^{l+1}-1-X$ (反碼)。

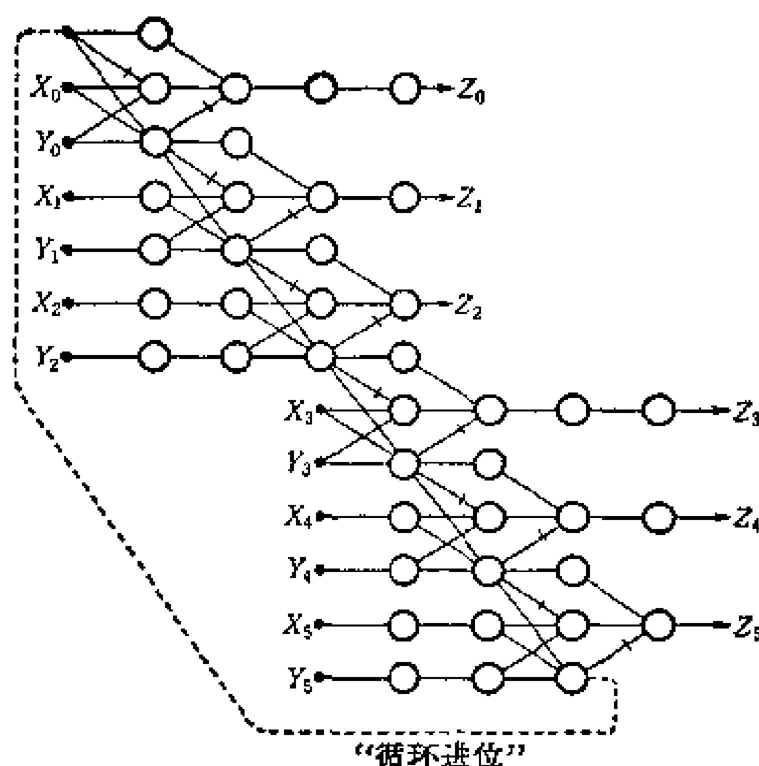


图42.1　并行二进加法器

后者是将 X 的各位均取"否定"而得，而前者除此以外还必須加个1❶。假如 $0<X<2^l$，則其补数 $2^l\leqslant X'<2^{l+1}$，并且最高位(2^l 位)的数字是1时表示是負数，这一位是**符号数字**。当使用"1的补数"(反碼)时，0有00…00和11…11二种表示，这要加以注意。

减法通常是以加上补数来执行的。使用"1的补数"(反碼)作 $X-Y$ 应由

$$X+2^{l+1}-1-Y$$

得到。这种計算中最高位(2^l)有进位出現，应当回送到最低一位(2^0)相加(循环进位 end around carry)，即 2^{l+1} 出現时加以舍弃，再加上 $2^0=1$，这就等于把 $2^{l+1}-1$ 减去了。把答案看作是 $(\mathrm{mod}\,(2^{l+1}-1))$，代替 $X-Y$ 而使用

$$X+2^{l+1}-1-Y$$

也是正确的。

❶ 前者常称为 X 的补碼，后者称为反碼。——譯者注

使用"2的补数"(补碼)时，把最高位(2^l)出現的进位完全舍弃。这种运算(mod 2^{l+1})中

$$X+2^{l+1}-Y$$

是和 $X-Y$ 同样的。但是，求"2的补数"(补碼)比較麻煩，因为向加法器輸入时，要将"1的补数"(反碼)加入，然后还要在"最低位上进位"即輸入处送入 $W_{-1}=1$.

(ii) **二进制加法綫路**(串行式) 用一个一位二进加法器(III_{10})，輸入二个串行的二进数 X, Y，进位輸出經过一个周期(通常是3級)的延迟，回送到一个輸入端(W)，在进位为1时与数字相加，则可得到串行加法器(图42.2). 将加法器的輸出送入延迟綫，延迟 l 后送入一个輸入端(例如 X 端)，就得到 **l 位的二进累加器**。

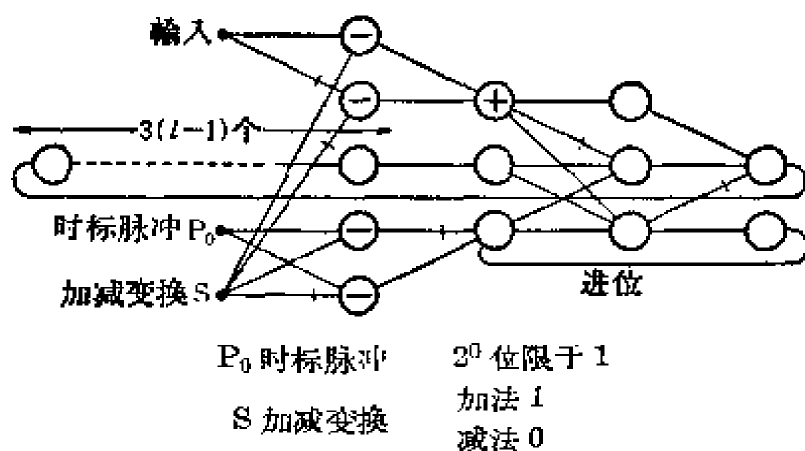
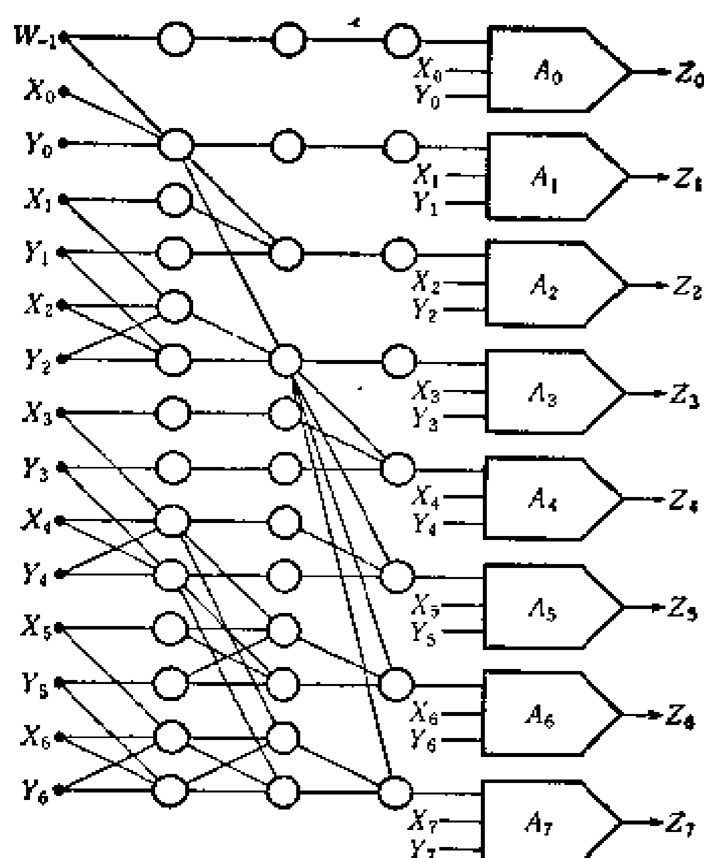


图42.2 串行累加器(加减法用"2的补数")

(iii) **二进加法器的高速进位綫路** 在图42.1的并行加法器中，若最低位所产生的进位要傳送到最高位(例如 111…11+00…01)須要很长的时間($l/3$ 周期)。虽然这种情况較少，但需等待这个最大的进位时間，再轉入下一个动作。

一种防止損失时間的方法是用称为"**进位檢查器**"的邏輯綫路，将进位信号消失的情况檢出，这样，平均运算时間显著地减少。但进位檢查器本身有延迟，因此，仍然影响下一动作的时間。



(A_0～A_7是1位加法綫路)

图 42.3　高速进位綫路的加法器

高速进位綫路是一种复杂的但也是最佳的綫路。这是先将各位的进位数 W_i 求出，再用他們算出各位的和(Z_i)的方法。求 W_i 的綫路是用三輸入端多数决定元件构成的快速綫路，本质上和图24.6的比較綫路相同。这个綫路的进位对于 ν 級是在 $2^\nu-1$ 位內进行的，因此位数增加，而时間不增加。所以在一定的时間內給出答案，即可进入下一个动作。

(iv) **十进制加法器**(并串型)　在十进制运算綫路中，特別是用变参元件綫路时，綫路中0，1是完全对应的"余3制"編碼，亦即用 $X+3$ 的二进数表示十进数 X 最为适宜。X 的补数$(9-X)$是

$$(9-X)+3=12-X=2^4-1-(X+3),$$

即得$(X+3)$的"1的补数"(反碼)，所以正負数对等地操作的运算綫路将呈現完全的自偶形式。

加法是

$$(X+3)+(Y+3)=(X+Y)+6,$$

当 $X+Y\geqslant 10$ 时，$X+Y+6\geqslant 2^4$，于是出現进位。

不出現进位时 $(X+Y+6)-3=(X+Y)+3$,

出現进位时 $(X+Y+6)-2^4+3=(X+Y-10)+3$,

从而得到正确的和数。然而，根据进位出現或不出現，加 3 或减 3 的**补正綫路**是必要的。补正綫路是簡单地在加数以外补以 ±3 的加法綫路。

并串型的构造上有一个問題，即 4 位的进位要在 4/3 周期内与上一位相加，这可由高速进位綫路来解决，图 42.4 是用高速进位的例子。这个綫路是不能抑制进位的，所以負数必須用"9 的补数"表示。

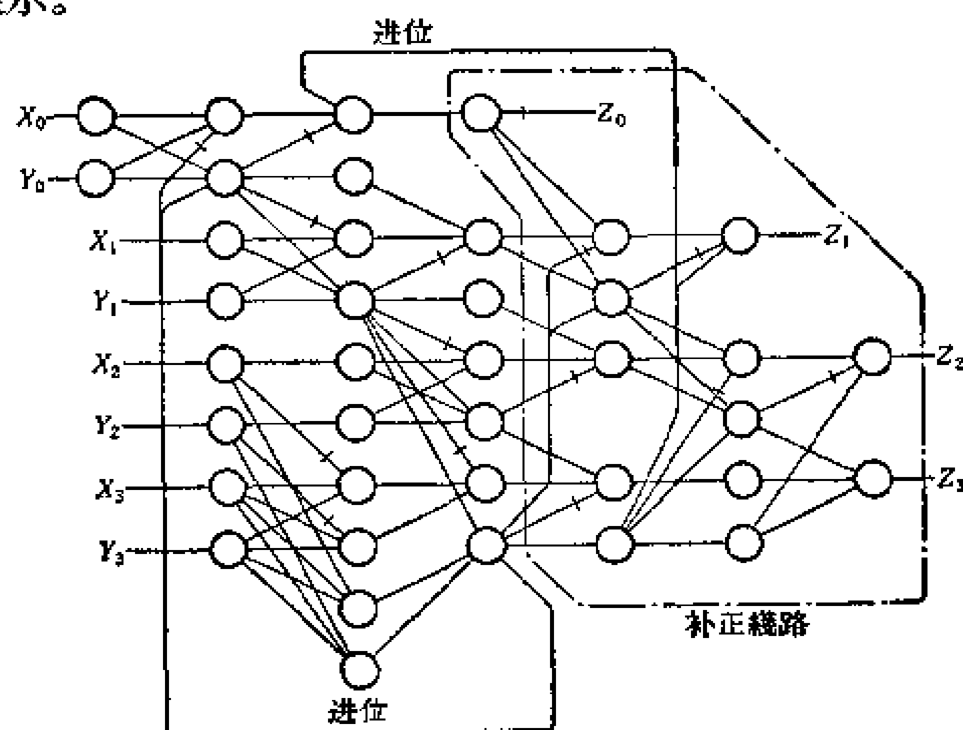


图 42.4 十进制加法器

(v) **二进制乘法綫路**(并行式) 二进制中乘法是将被乘数的 1 倍，2 倍，2^2 倍，… 相加，乘数的各位数字控制加法的执行，并且被乘数要逐步左移，通过乘数的各位数字控制的門向累加器相

加。实际上被乘数幷不动，相反，是将累加器中的部分和向右移。

图 42.5 是以幷行加法器实行这种操作的綫路。进位延迟一周期后，恰巧在同一位上和右移一位后的上位的部分和相加。进位不必一次实行，随着計算的进行，一步一步地实行，綫路可簡单了。所以，經过 l 位 l 次加法，再将进位整理完毕，就得到了正确的答案。通常一次乘法要 $2l$ 个周期的时間。

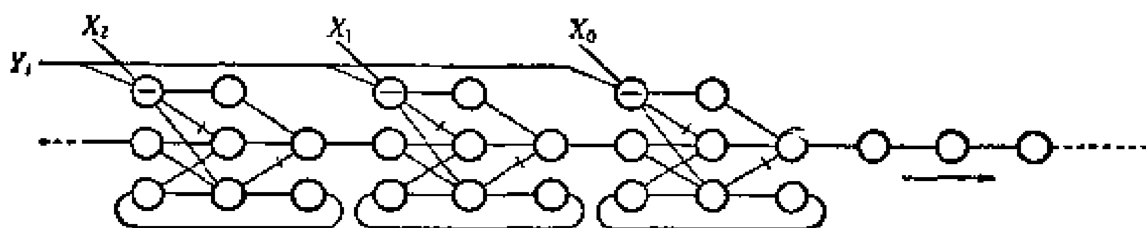


图 42.5　幷行乘法器

(vi) **二进制乘法綫路**(串行式)　用串行加法器实行乘法，l 次加法每次要 l 个周期，总的要 l^2 周期。然而乘数的有效位数少时可以使时間縮短，所以一般时間比它短。

这里所示的乘法器，用了一个稍微特殊的方法。它的特征是无論乘数的正負都同样簡单，这是基于下述原理：l 位(不包含符号)的数 $X=\sum_{r=0}^{l-1} 2^r X_r - 2^l X_l$，即

$$X=\sum_{r=0}^{l-1}(2^{r+1}-2^r)X_r-2^lX_l=\sum_{r=0}^{l}2^r(X_{r-1}-X_r)\quad (X_{-1}=0),$$

则

$$XY=\sum_{r=0}^{l}2^r(X_{r-1}-X_r)Y.$$

将 Y 左移，調查 X 的各位数字，若 X_r 与 X_{r-1} 相同，則什么也不做。若 $X_{r-1}=0$，$X_r=1$，則加 -2^rY。若 $X_{r-1}=1$，$X_r=0$，則加 2^rY。

图 42.6 是实行这个方法的綫路，綫路中 $FF1$ 置 1 时开始乘法操作。将乘数的数字由低位順序調查，如果最初是 1 出現，$FF2$

置"1"并开始执行减法，同时計数器动作，当 l 位减法做完，計数器給出信号，$FF2$ 还原，减法結束。如果后面出現 0，$FF2$ 又置 1，这时开始做加法，如此，加减法交替进行。当进入 l 位时，則 $FF1$ 还原，运算結束。乘数中 0 和 1 交变一次，就要执行运算一次。所以不管数的正負，有效数字少的乘法时間就短。

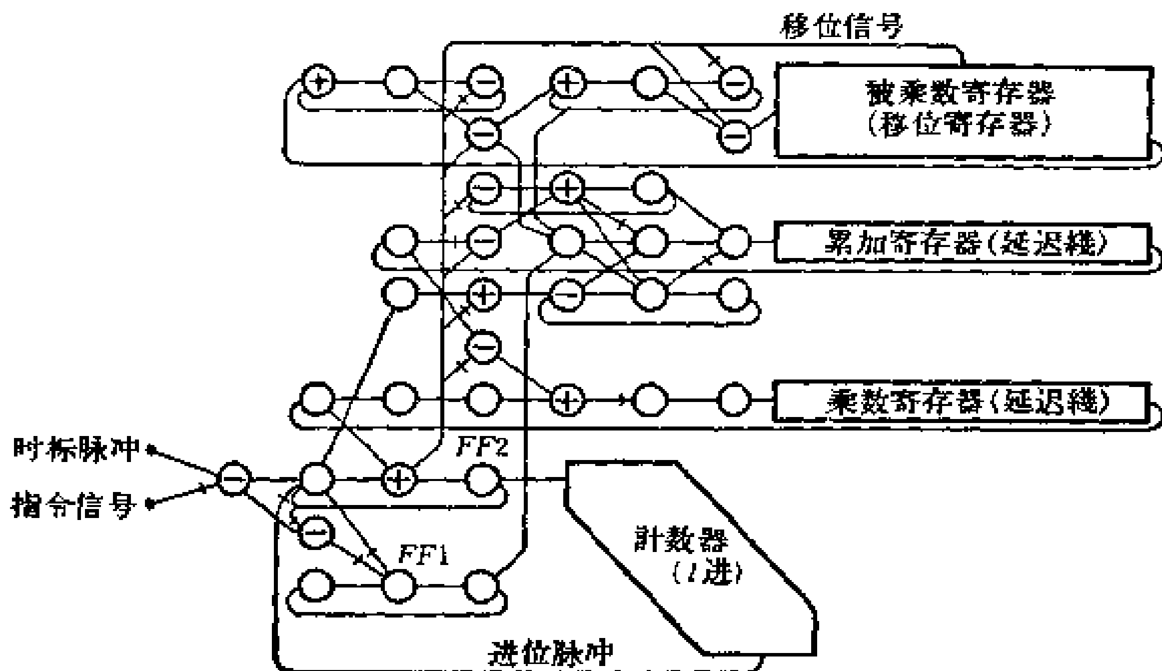


图 42.6 串行乘法器

为了实行加法和减法的交变动作起見，可在累加器环路中輸入否定，每执行一周后即把符号反轉而 Y 不变。同样乘数寄存器也每次反轉，可簡单地檢查 0→1，1→0 的变化。被乘数寄存器是一个移位寄存器，在探查乘数变化时应保持静止，作短时間的等待。

(vii) **十进制乘法綫路** 十进制乘法有下列几种：

(a) 加法的重复(台式計算机的做法)。

(b) 由九九表組成。

(c) 用 2 倍、4 倍或 2 倍、5 倍的綫路作加法。

(d) 将 2～9 倍的被乘数算出，再选取相加。

这里不作詳述。

(viii) **二进制除法綫路**　二进制的除法方法本質上和台式計算机是一样的。从被除数中减去除数，在二进制中，因为减算次数不是0，就是1，若剩余比除数小，则将除数移位，若剩余比除数大，則减去一次除数再移位。

实际是，不論大小，都作一次减法，若其結果为負，再加回去，商数記0；若减算的結果为正，則商数記1.

这称为**加回式**(restoring method)，但减算后还要加回，运算次数較多。另一种**不加回式**(non-restoring method)❶，是当結果为負时，商数記 −1，下一次作加法；若結果为正，則商数記1，下次作减法。

其答案如下(設被除数及除数均为正数)：

$$Z=2^l+\sum_{r=1}^{l}2^{r-1}\bar{Z}_r,$$

其中 $\bar{Z}_r=\pm1$. 假設 $\bar{Z}_r=2Z_r-1$，則得

$$Z=2^l+\sum_{r=1}^{l}2^rZ_r+\sum_{r=1}^{l}2^{r-1}=\sum_{r=1}^{l}2^rZ_r+1.$$

当然，每次的剩余为正时，則商1；为負时則商0. 最后若商1(对应于余数为正)，則得正确的商数。图42.7是实行这种方法的框图。

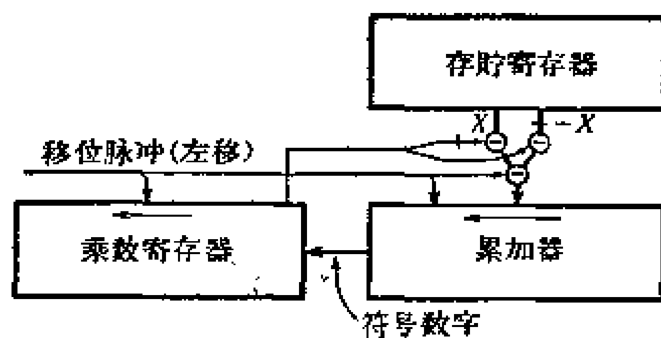


图42.7　除法的方法(不加回式)

❶ 通常称正負1制除法。——譯者注